UNICHECK

Власник документу:
Попенко Володимир Дмитрович

ID перевірки:
1004035889

Дата перевірки:
15.06.2020 03:45:43 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
15.06.2020 03:50:46 EEST

ID користувача:
77149

Назва документу: Zozulja_ip63 (1)

ID файлу: 1004048952 Кількість сторінок: 48 Кількість слів: 7368 Кількість символів: 58552 Розмір файлу: 93.24 KB

2.82% Схожість

Найбільша схожість: 1.62% з джерело бібліотеки. ID файлу: 1000020564

0.72% Схожість з Інтернет джерелами	15	Page 50
2.82% Текстові збіги по Бібліотеці акаунту	61	Page 50

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів	1

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)
Автоматизованих систем обробки інформації і управління
(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему: «*Ігрове застосування з навчанням ботів-суперників з використанням нейронних мереж»*

Виконав: студент IV курсу, групи *ІП-63 Зозуля Андрій Вячеславович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст.в., Ковтунець О.В.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант з графічної документації *доц., Ліщук К.І.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент *доц., Пасько В.П.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Зозулі Андрію Вячеславовичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Ігрове застосування з навчанням ботів-суперників з використанням нейронних мереж»*
керівник проєкту *Ковтунець Олесь Володимирович, ст.в.*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджена наказом по університету від “07” травня 2020 р. №1081-с
2. Термін подання студентом проєкту «08» *червня 2020 року*
3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення і терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення
3) Аналіз якості програмного забезпечення та опис випробувань
4) Керівництво користувача

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

1) Схема структурна варіантів взаємодії

2) Схема структурна компонентів програмного забезпечення

3) Схема структурна класів програмного забезпечення

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*10.03.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*15.03.2020*	
3.	*Постановка та формалізація задачі*	*20.03.2020*	
4.	*Аналіз вимог до програмного забезпечення*	*25.03.2020*	
5.	*Алгоритмізація задачі*	*30.03.2020*	
6.	*Моделювання програмного забезпечення*	*04.04.2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*10.04.2020*	
8.	*Розробка архітектури програмного забезпечення*	*15.04.2020*	
9.	*Розробка програмного забезпечення*	*20.04.2020*	
10.	*Налагодження програми*	*22.04.2020*	
11.	*Виконання графічних документів*	*23.04.2020*	
12.	*Оформлення пояснювальної записки*	*01.05.2020*	
13.	*Подання ДП на попередній захист*	*15.05.2020*	
14.	*Подання ДП рецензенту*	*01.06.2020*	
15.	*Подання ДП на основний захист*	*08.06.2020*	

Студент _____ Андрій Зозуля
(підпис)

Керівник _____ Олесь Ковтунець
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	A4		Завдання на дипломний проєкт	2	
2	A4	КПІ.ІП-6312.045490.02.81	Пояснювальна записка	66	
3	A4	КПІ.ІП-6312.045490.03.91	Технічне завдання	10	
4	A4	КПІ.ІП-6312.045490.04.13	Опис програми	23	
5	A4	КПІ.ІП-6312.045490.05.51	Програма та методика тестування	6	
6	A4	КПІ.ІП-6312.045490.06.34	Керівництво користувача	6	
7	A3	КПІ.ІП-6312.045490.07.99.СС	Схема структурна варіантів використань	1	
8	A3	КПІ.ІП-6312.045490.07.99.СС	Схема структурна компонентів програмного забезпечення	1	
9	A3	КПІ.ІП-6312.045490.07.99.СС	Схема структурна класів програмного забезпечення	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6312.045490.00.90		
Розробн.		Зозуля А.В.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Ковтунець О.В.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-63	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: «Ігрове застосування з навчанням ботів-суперників з використанням нейронних мереж»

Київ – 2020 року

АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з чотирьох розділів, містить 41 таблиць, 16 рисунків та 10 джерел — загалом 116 сторінок.

Об'єкт дослідження: машинне навчання ботів-суперників з використанням нейронних мереж.

Мета дипломного проекту: інтеграція машинного навчання моделі ботасуперника з використанням нейронної мережі в ігрове застосування.

У першому розділі виконано аналіз предметної області, відомих технічних рішень, сформульовано функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення.

У другому розділі розроблена архітектура ігрового застосування, побудовано модель навчання бота-суперника та проаналізована структура вхідних та вихідних даних моделі нейронної мережі.

У третьому розділі проведено аналіз якості програмного забезпечення, та наведено тестовий приклад.

У четвертому розділі описано керівництво користувача.

КЛЮЧОВІ СЛОВА: ІГРОВІ ЗАСТОСУВАННЯ, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ.

ABSTRACT

Explanatory note of the diploma project consists of four sections, containing 41 tables, 16 figures and 10 sources - total 116 pages.

The object of research: machine learning of the bot-opponent with use of neural networks.

The aim of the project: integration of machine learning the model of the bot-opponent using a neural network in game application.

In the first section were analyzed subject area and known technical solutions, formulated functional and non-functional requirements for the software developed.

In the second section were developed game application architecture contstructed training model of bot-opponent and analyzes the structure of input and output data of neural network model.

The third section analyzes the software quality and provides a test case.

The fourth section describes user manual.

KEY WORDS: GAME APPLICATIONS, MACHINE LEARNING, NEURAL NETWORKS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Reinforcement learning (укр. навчання з підкріпленням) — це галузь машинного навчання, яка описує які дії повинні виконувати програмні агенти в певному середовищі задля максимізації деякого уявлення про сукупну винагороду.

Unity — багатоплатформовий ігровий рушій для розробки дво- та тривимірних додатків та ігор.

The Unity Machine Learning Agents Toolkit (ML-Agents) — це проект з відкритим кодом, який дозволяє інтегрувати в ігрове середовище навчання інтелектуальних агентів на основі моделі Reinforcement learning.

ПЗ — Програмне забезпечення.

ВСТУП

З розвитком комп'ютерних технологій зросла здатність обчислювальних пристроїв швидко обробляти великі обсяги даних, що, дозволяє розробникам програмного забезпечення використовувати все більш складні алгоритми навіть у невеликих програмах.

Нові можливості апаратних засобів провокує стрімкий ріст всіх галузей сфери інформаційних технологій, розробку нових підходів, індустрій та способів використання обчислювальних пристроїв в житті людей. Одним з яскравих прикладів такого розвитку є індустрія комп’ютерних ігор, що зародилась в середині 1970-х років. За даними аналітичної компанії Newzoo, світовий прибуток індустрії комп’ютерних ігор в 2019 році сягає 148,8 мільярда доларів США з прогнозованим ростом в +7,2%.

Такий стрімкий економічний розвиток індустрії, в поєднанні з не менш стрімким розвитком потужності обчислювальної техніки, дає змогу інтегрувати інноваційні технології для реалізації нових підходів, на які раніше не вистачало ресурсу обчислювальних потужностей.

Одним з таких підходів є використання елементів штучного інтелекту. Інтеграція елементів машинного навчання суттєво підвищує зацікавленість гравця до ігрового продукту, адже “розумний” бот-суперник може кинути виклик будь-якому реальному гравцю та мотивувати його на розвиток здібностей, розробку нових стратегій гри та підходів до подолання перешкод.

Таким чином, створення ігрових застосунків із введенням до них елементів штучного інтелекту зумовлює актуальність даної роботи.

Об’єктом дослідження даного проекту є машинне навчання ботів-суперників з використанням нейронних мереж в сфері розробки комп’ютерних ігор.

Мета даної роботи – інтеграція машинного навчання моделі бота-суперника з використанням нейронної мережі в ігрове застосування.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Розробка ігрових застосунків в наш час досягла світового рівня та є однією з найбільш широких галузей в сфері інформаційних технологій. Розвиток даної галузі є один з найвизначніших каталізаторів розвитку сфери обчислювальної техніки. Відомі світові бренди, які виготовляють комплектуючі для обчислювальних машин, такі як: Nvidia, Intel, AMD, Sony, Microsoft та інші, в значній мірі залежать від тенденцій розвитку ігрової індустрії. Гравці та розробники потребують все більш потужні обчислювальні машини для гри та розробки ігор відповідно.

Таким чином, розвиток в цій галузі та її тенденції значним чином впливають на окремі напрямки розвитку сфери інформаційних технологій та її економіку в цілому. Такий вплив потребує значних зрушень в підходах до проектування і розробки відеоігор.

Дана індустрія поділяється на велику кількість напрямків. Від простих 2D ігор для мобільних пристроїв, з нескладними механіками та сценаріями, до масштабних проектів ААА класу, що потребують вагомих фінансових інвестицій, команд розробників та інших складових. Проте, всіх їх об’єднує одна, безперечно, найбільш значуща ціль – зацікавити свого гравця та привернути увагу нових потенційних гравців.

Для досягнення даної цілі використовують різні підходи: цікаві сюжетні сценарії, використання якісного та привабливого контенту, унікальні механіки гри та інші. З розвитком області машинного навчання, розробники ігор отримали потужний та гнучкий інструмент для урізноманітнення ігрових механік унікальними сценаріями поведінки агентів, що інтегровані в ігрове середовище. Саме навчання ігрових ботів дало змогу розробникам створити унікальні механіки, які можуть зацікавити гравця з будь-яким смаком.

1.2 Змістовний опис і аналіз предметної області

1.2.1 Проектування ігрових застосунків

Процес створення ігрових застосунків, як і будь-якого ПЗ, починається з проектування. До переліку питань, які визначаються на етапі проектування відносять наступні питання:

Основна ігрова концепція

Це питання є ключовим в подальшому плануванні, так як безпосередньо впливає на вектор розвитку проекту. На даному етапі визначаються ключові складові майбутньої гри. Узгоджується як має виглядати майбутня гра, в якому жанрі буде організовано основні механіки, для яких платформ буде вестись розробка, яким чином буде заохочуватись гравець до подальшої гри. На даному етапі немає необхідності в деталях описувати як будуть відбуватись чи реалізовуватись ті чи інші складові, найважливіше – загальне уявлення. Правильний опис і планування основної концепції – запорука успішного подальшого розвитку проекту, так як кардинальна зміна концепції готового проекту складніша за перепроектування і розробку нового. В той же час, опис концепції повинен бути лаконічним і зрозумілим, щоб зацікавити потенційного інвестора, не заплутати розробника чи не дати можливості двояко трактувати ідею будь-кому, хто з ним ознайомиться. Для цього концепт не повинен містити незрозумілу технічну інформацію, деталі реалізації, проектні рішення, тощо.

Перелік основних механік з детальним описом

Після формалізації і опису ідеї в основній ігровій концепції, розробляють детальний опис основних ігрових механік. На даному етапі створюється повний перелік всіх можливих механік взаємодії майбутнього гравця з грою.

До основних механік можна віднести:

– взаємодію з графічним інтерфейсом користувача;
– основну ігрову механіку, або їх перелік, якщо такий наявний;
– механіку взаємодії з іншими гравцями, якщо така наявна;
– механіку взаємодії з штучними гравцями, або сутностями;

– та інші.

Ігровий сценарій

Наступним кроком є опис ігрового сценарію. Можна виокремити два типи сценаріїв:

– локальний сценарій сцени, який описує сценарій того, що буде відбуватись на кожній конкретній сцені (ігрова сцена, сцена вікна налаштувань, стартова сцена, тощо);

– сюжетний сценарій, що описує сюжетну лінію гри, якщо вона передбачена основною концепцією.

В описі сценарію, як і в формулюванні концепції варто уникати використання двозначних термінів та формулювань. Проте, опис сценарію повинен містити якомога детальніший перелік подій, які відбуватимуться, і їх детальний опис. Такий підхід до формування ігрового сценарію допоможе підчас майбутньої розробки максимально чітко уявляти кінцевий вигляд ігрового застосування на кожному етапі розробки. Це допоможе уникнути локальних неточностей та мінімізує кількість правок після реалізації гри.

Стилістика ігрового контенту

Безумовно, дизайн та зовнішній вигляд графічного інтерфейсу є важливим компонентом будь-якого програмного забезпечення, що створене для пересічного користувача. Звичайно, на функціональні властивості це майже не впливає, проте, в індустрії комп’ютерних ігор саме якість графічного матеріалу, його стилістика і відповідність нормам популярних течій ігрової індустрії відіграє рішучу роль в кінцевій успішності проекту. Зазвичай, формується банк рисунків, який визначає особливості, дизайнерські рішення та основну стилістичну концепцію. Такі банки складаються художниками разом з ігровими дизайнерами та є базою, на яку опиратимуться при подальшій розробці контенту для наповнення гри.

Результатом планування ігрового застосунку є документація, що поділяється на дві частини. Перша – опис майбутньої гри як кінцевого бізнес-продукту в розгорнутому вигляді. Друга - документ, який містить опис початкового опрацювання всіх аспектів та елементів гри.

Роль даної документації важко переоцінити, так як відповідно до неї всі учасники розробки матимуть можливість в повній мірі ознайомитись з деталями майбутньої розробки та сформулювати повноцінне уявлення про всі етапи реалізації. А отже, на всіх рівнях та етапах розробки буде поставлено чіткі задачі згідно з заявленою документацією. Для ігрових дизайнерів ця документація є місцем збереження та висловлення концептуальних ідей. Розробник має чіткий опис того, що він має зробити, а це дає можливість завчасно побудувати потрібну архітектуру застосунку та уникнути подальших проблем з масштабуванням або корекцією вже готової архітектури. Менеджери проекту мають змогу чітко розпланувати етапи розробки та проводити контроль за виконанням завдань. Також, на початкових етапах, така документація дає змогу ефективніше залучати зовнішні інвестиції, так як майбутній інвестор матиме змогу ознайомитись ретельно з деталями проекту і оцінити його рентабельність.

Важливо, щоб вся дизайнерська та продуктова документація оновлювалася протягом всього періоду розробки проекту.

Для її ефективного оновлення та застосування проектної документації, необхідно звернутись до використання спеціальних інструментів, які суттєво підвищують ефективність процесу паралельного редагування декількома учасниками розробки, а також дозволяють всім членам команди якнайшвидше отримувати необхідну актуальну інформацію, про продукт і всіх його змін.

Дана проектна документація буде основою для наступного етапу розробки – імплементації ігрового застосунку.

1.2.2 Імплементація ігрових застосунків

Імплементація ігрового застосунку – це процес розробки архітектури ігрового застосунку, написання програмних кодів, що реалізують розроблену архітектуру та наповнення гри контентом.

Згідно з проектною документацією, командою розробників формується програмна архітектура. На етапі розробки архітектури визначаються ключові деталі програмної реалізації, а саме:

– мова програмування, якою буде вестись розробка;

– ігровий рушій, який буде взято за основу для реалізації інтерфейсної частини застосунку. Саме вибір рушія часто впливає на кінцевий вигляд архітектури додатку та його програмну реалізацію. Серед відомих ігрових рушіїв можна виділити: Unity, Unreal Engine та Cocoa. Всі вони мають характерні особливості, що можуть вплинути на процес розробки;

– структура збереження даних користувачів;

– перелік бібліотек та фреймворків, що будуть застосовані для програмної реалізації продукту.

Згідно з розробленою архітектурою починається процес імплементації кодів програмного забезпечення.

Після імплементації програмних кодів, гра заповнюється готовим ігровим контеном. Сюди можна віднести реалізацію ігрових сценарії, що описані в проектній документації, додавання попередньо намальованого арту, звуків, тощо. Даний процес супроводжується тісною співпрацею з ігровими дизайнерами, менеджерами проекту, художниками, звуковими сценаристами та аніматорами.

1.2.3 Тестування ігрових застосунків

Готовий продукт проходить тестування згідно з проектною документацію на відповідність до запланованого вигляду продукту, функціонально стійкість розробленого ПЗ та відповідність до вимог, що висуваються для ігрового продукту. Проводиться аналіз на придатність для використання гравцем.

В ході тестування збираються аналітичні дані по основних метриках про поводження реалізованого застосунку.

Виявлені недоліки відправляються на доопрацювання розробниками згідно з специфікою кожного окремого недоліку.

1.2.4 Інтеграція машинного навчання в ігрові середовища

Розвиток галузі машинного навчання в наш час дає широкі можливості для інтеграції в ігрові програмні продукти елементів штучного інтелекту.

Найбільш придатним для цього є метод машинного навчання Reinforcement Learning.

Суть методу полягає в інтеграції агента з моделлю нейронної мережі в середовище, з яким він буде взаємодіяти, і нагороджувати його за досягнення тих чи інших цілей позитивною або негативною нагородою відповідно.

Для інтеграції нейронної мережі в агента використовують одну з поширених відкритих бібліотек, таких як: NumPy, Pandas, Matplotlib, Tensorflow, Keras та інші.

Важливим є побудова правильної архітектури нейронної мережі та моделі винагороджування агента. Одним з головних правил правильної побудови моделі винагороджування агента є надавання винагороди за досягнення певних цілей, а не за виконання кроків, які на думку розробника можуть призвести до досягнення поставлених цілей. Не менш важливим є правильне надання даних до вхідного шару нейронної мережі. Основною рекомендацією є нормалізація вхідних параметрів для представлення на вхід в мережу. Це допоможе уникнути непередбачуваної поведінки агента при тестуванні на тестувальній вибірці, що може відрізнятись від тренувальної вибірки.

1.3 Аналіз успішних ІТ-проектів

1.3.1 Аналіз відомих технічних рішень

Серед технічних рішень, що користуються популярністю в сфері інтеграції машинного навчання в ігрові застосування, привертає увагу відкритий пакет інструментів ML Agents від компанії Unity Technologies.

Даний пакет надає широкий спектр інструментів для інтеграції методу машинного навчання Reinforcement learning в середовище ігрової розробки Unity. Поєднання цих двох потужних інструментів дає високу гнучкість в побудові власних моделей, інтегрованих в ігрове середовище, що розроблене за допомогою рушія Unity.

Беззаперечними перевагами даного технічного рішення є:

– гнучкість в розробці ігрового середовища;

– широкий спектр інструментів для навчання власних моделей штучного інтелекту;

– можливість проведення навчання на дистанційних платформах;

– доступність методичних матеріалів та інструкцій для користувача;

– підтримка всіх популярних платформ та пристроїв;

– простота в інтеграції навченої моделі в ігрове застосування.

1.3.2 Аналіз відомих програмних продуктів

Ринок ігрової індустрії представлений великою кількістю успішних проектів, що мають світовий успіх.

Серед відомих ІТ-проектів в ігровій індустрії, з інтеграцією ботів-суперників, реалізованих з використанням машинного навчання, можна віднести наступні популярні ігрові розробки:

Quake 3 Arena

В 2018 році компанія DeepMind опублікувала свої досягнення в сфері інтеграції ботів-суперників з використанням машинного навчання для даного продукту. Для навчання було використано метод навчання з підкріпленням, що

дало змогу навчити штучний інтелект пристосовуватись до динамічних змін стану ігрового середовища і продемонструвати високий рівень натренованості, що дало змогу виграти реальних гравців в більшості ігрових партій

Перевагами даної розробки є:

– високий рівень натренованості агентів;
– ефективна взаємодія агентів з ігровим середовищем;
– ефективна взаємодія агентів з реальними гравцями;
– ефективна взаємодія агентів з іншими агентами.

Серед недоліків можна виділити високу витрату агентами обчислювальних ресурсів, що не дає змоги застосовувати їх на платформах з низькою потужністю обчислень.

Super Mario Bros.

Популярна у всьому світі гра, яка в нових версіях була доповнена генерацією рівнів з використанням нейронних мереж.

До основних переваг даного продукту відносяться:

– нетривіальні сценарії генерації ігрових рівнів;
– низький рівень використання обчислювальних ресурсів натренованою моделлю.

Головним недоліком даного продукту є потенційно непередбачувана поведінка моделі, що в деяких випадках призводить до генерації рівнів, що неможливо пройти людині. Даний недолік виявлений самими ж розробниками, підчас тестування штучним гравцем, проте, підчас тестування реальними гравцями, таких випадків зафіксовано не було.

1.4 Аналіз вимог до програмного забезпечення

Вимоги до програмного продукту ставляться зважаючи на три основні фактори: який функціонал реалізовує програмний продукт, хто потребує такий функціонал, та яка в ньому необхідність. Таким чином для визначення вимог по програмного забезпечення потрібно виділити ролі користувачів, функціональні вимоги, які відносяться до цих ролей та пояснення до цих вимог. Окремо

ставляться нефункціональні вимоги, в яких описуються характеристики результуючого продукту, та певні обмеження на те, як має бути реалізований функціонал.

Оскільки результуючим продуктом є ігрове застосування, можна виділити лише одну роль користувача — гравець (далі Гравець).

Гравець має мати змогу користуватись ігровим застосуванням без додаткових інсталяцій.

1.4.1 Розроблення функціональних вимог

Для повноти функціональних вимог ігрового застосування потрібно розглянути всі сценарії взаємодії Гравця з ігровим застосуванням. Дані сценарії представлені у вигляді варіантів взаємодії Гравця та програмного продукту.

Передбачені варіанти взаємодії представлені в наступних таблицях:

Таблиця 1.1 – Варіант взаємодії номер UC001

Номер	UC001
Назва	Запуск ігрового застосування
Опис	Гравець має можливість запустити ігрове застосування на своєму пристрої
Учасники	Гравець
Постумови	Гравець отримує початковий екран ігрового застосування
Основний сценарій	Гравець запускає ігрове застосування на своєму пристрої; Ігрове застосування відображає стартову сцену гри
Розширення сценаріїв	У випадку, коли ігрове застосування встановлене неправильно, Гравець повинен отримати повідомлення про помилку інсталяції ігрового застосування

Таблиця 1.2 – Варіант взаємодії номер UC002

Номер	UC002
Назва	Старт головної ігрової сцени
Опис	Гравець має можливість запустити головну ігрову сцену
Учасники	Гравець
Постумови	Ігрове застосування відображає головну ігрову сцену
Основний сценарій	Гравець натискає на кнопку старту гри в центральній частині стартової сцени
Розширення сценаріїв	

Таблиця 1.3 – Варіант взаємодії номер UC003

Номер	UC003
Назва	Відкриття інтерфейсу налаштувань гри з стартової сцени гри
Опис	Гравець має можливість відкрити інтерфейс налаштувань гри
Учасники	Гравець
Постумови	Ігрове застосування відображає інтерфейс налаштувань гри
Основний сценарій	Гравець натискає на кнопку налаштувань в центральній частині
Розширення сценаріїв	

Таблиця 1.4 – Варіант взаємодії номер UC004

Номер	UC004
Назва	Взаємодія гравця з ігровим середовищем
Опис	Гравець має можливість взаємодіяти з ігровим середовищем шляхом керування персонажем гравця

Продовження таблиці 1.4

Учасники	Гравець
Постумови	Гравець отримує результати взаємодії з ігровим середовищем
Основний сценарій	Гравець натискає на клавіші «W», «S», «A», «D», або їх відповідні аналоги, свого пристрою потокового введення для пересування свого персонажа в ігровому середовищі вперед, назад, ліворуч та праворуч відповідно
Розширення сценаріїв	

Таблиця 1.5 – Варіант взаємодії номер UC005

Номер	UC005
Назва	Відкриття вікна паузи гри з головної ігрової сцени
Опис	Гравець має можливість відкрити вікно паузи з головної ігрової сцени
Учасники	Гравець
Постумови	Ігрове застосування ставить на паузу головну ігрову сцену та відображає вікно паузи
Основний сценарій	Гравець натискає кнопку паузи у верхньому правому кутку головної ігрової сцени
Розширення сценаріїв	

Таблиця 1.6 – Варіант взаємодії номер UC006

Номер	UC006
Назва	Вихід з вікна паузи на головну ігрову сцену
Опис	Гравець має можливість продовжити взаємодію з ігровим середовищем шляхом виходу з меню паузи на головну ігрову сцену

Продовження таблиці 1.6

Учасники	Гравець
Постумови	Ігрове застосування закриває вікно паузи та відновлює роботу головної ігрової сцени
Основний сценарій	Гравець натискає на кнопку продовження гри в верхній частині вікна паузи
Розширення сценаріїв	

Таблиця 1.7 – Варіант взаємодії номер UC007

Номер	UC007
Назва	Припинення роботи ігрового застосування з стартової сцени
Опис	Гравець має можливість припинити роботу ігрового застосування з стартової сцени гри
Учасники	Гравець
Постумови	Ігрове застосування зупиняє свою роботу шляхом закривання
Основний сценарій	Гравець натискає на кнопку закриття ігрового застосування в лівій частині стартової сцени
Розширення сценаріїв	

Таблиця 1.8 – Варіант взаємодії номер UC008

Номер	UC008
Назва	Припинення роботи ігрового застосування з вікна меню
Опис	Гравець має можливість припинити роботу ігрового застосування з вікна меню
Учасники	Гравець
Постумови	Ігрове застосування зупиняє свою роботу шляхом закривання

Продовження таблиці 1.8

Основний сценарій	Гравець натискає на кнопку закриття ігрового застосування в нижній частині вікна меню
Розширення сценаріїв	

Таблиця 1.9 – Варіант взаємодії номер UC009

Номер	UC009
Назва	Перезавантаження головної ігрової сцени з вікна меню
Опис	Гравець має можливість перезавантажити головну ігрову сцену з вікна меню
Учасники	Гравець
Постумови	Ігрове застосування закриває вікно меню та перезавантажує головну ігрову сцену
Основний сценарій	Гравець натискає на кнопку перезавантаження ігрової сцени в правій частині вікна меню
Розширення сценаріїв	

Таблиця 1.10 – Варіант взаємодії номер UC010

Номер	UC010
Назва	Відкриття інтерфейсу налаштувань гри з вікна меню
Опис	Гравець має можливість відкрити інтерфейс налаштувань гри з вікна меню
Учасники	Гравець
Постумови	Ігрове застосування закриває вікно меню та відображає інтерфейс налаштувань гри
Основний сценарій	Гравець натискає на кнопку налаштувань в лівій частині вікна меню
Розширення сценаріїв	

Таблиця 1.11 – Варіант взаємодії номер UC011

Номер	UC011
Назва	Взаємодія Гравця і бота-суперника
Опис	Гравець має можливість взаємодіяти з ботом-суперником в межах ігрового середовища
Учасники	Гравець
Постумови	Гравець отримує результат взаємодії з ботом-суперником шляхом колізії ігрового персонажа Гравця та ігрового персонажа бота-суперника; Колізія двох персонажів означає програш Гравцем ігрового рівня; Програш рівня фіксується та надається можливість перезавантажити головну ігрову сцену для нового старту рівня
Основний сценарій	Гравець контролює пересування свого ігрового персонажа в ігровому середовищі шляхом, що передбачено варіантом взаємодії номер UC004
Розширення сценаріїв	

На основі описаних варіантів взаємодії гравця та ігрового застосування можна виділити наступні функціональні вимоги до розроблюваного програмного продукту, що описані в наступних таблицях:

Таблиця 1.12 – Функціональна вимога номер REQ001

Номер	REQ001
Назва	Запуск ігрового застосування
Опис	Ігрове застосування повинно коректно запускатись після правильного встановлення на пристрої Гравця

Таблиця 1.13 – Функціональна вимога номер REQ002

Номер	REQ002
Назва	Стартова сцена
Опис	Ігрове застосування повинно запускати стартову сцену після свого запуску

Таблиця 1.14 – Функціональна вимога номер REQ003

Номер	REQ003
Назва	Інтерфейс налаштування гри
Опис	Ігрове застосування повинно мати інтерфейс налаштування гри та відривати його на вимогу користувача з стартової сцени або вікна паузи

Таблиця 1.15 – Функціональна вимога номер REQ004

Номер	REQ004
Назва	Вікно меню
Опис	Ігрове застосування повинно мати вікно паузи та відкривато його на вимогу користувача з головної ігрової сцени

Таблиця 1.16 – Функціональна вимога номер REQ005

Номер	REQ005
Назва	Взаємодія з ігровим середовищем
Опис	Ігрове застосування повинно надавати гравцю можливість взаємодіяти з ігровим середовищем через потоковий пристрій введення

Таблиця 1.17 – Функціональна вимога номер REQ006

Номер	REQ006
Назва	Зупинка роботи ігрового застосування
Опис	Ігрове застосування повинно коректно зупиняти свою роботу на вимогу користувача з стартової сцени або вікна паузи

Таблиця 1.18 – Функціональна вимога номер REQ007

Номер	REQ007
Назва	Призупинення оновлення головної сцени
Опис	Ігрове середовище має коректно призупиняти взаємодію Гравця та бота-суперника з ігровим середовищем, та один з одним

Таблиця 1.19 – Функціональна вимога номер REQ008

Номер	REQ008
Назва	Інтерфейс вікна меню
Опис	Інтерфейс вікна паузи повинен давати Гравцю доступ до: продовження взаємодії з ігровим середовищем головної ігрової сцени, закриття ігрового застосування, перезавантаження головної ігрової сцени та відкриття інтерфейсу налаштувань гри

Таблиця 1.20 – Функціональна вимога номер REQ009

Номер	REQ009
Назва	Інтерфейс взаємодії гравця з ігровим середовищем
Опис	Ігрове застосування повинно зчитувати сигнали з клавіш «W», «S», «A», «D», або їх відповідні аналогів, з пристрою потокового введення Гравця. Дані сигнали є керуючими для ігрового персонажа Гравця

Таблиця 1.21 – Функціональна вимога номер REQ010

Номер	REQ010
Назва	Бот-суперник
Опис	Ігрове застосування повинно генерувати в протидію Гравцю бота-суперника з навченою моделлю на основі нейронної мережі

Таблиця 1.22 – Функціональна вимога номер REQ011

Номер	REQ011
Назва	Взаємодія гравця з ботом-суперником

Продовження таблиці 1.22

Опис	Ігрове застосування повинно оброблювати колізії персонажів гравця та бота-суперника

Покриття функціональними вимогами до ігрового застосування з навчанням ігрових ботів з використанням нейронних мереж варіантів взаємодії зображено на рисунку 1.1 – Матриця покриття функціональними вимогами варіантів взаємодії.

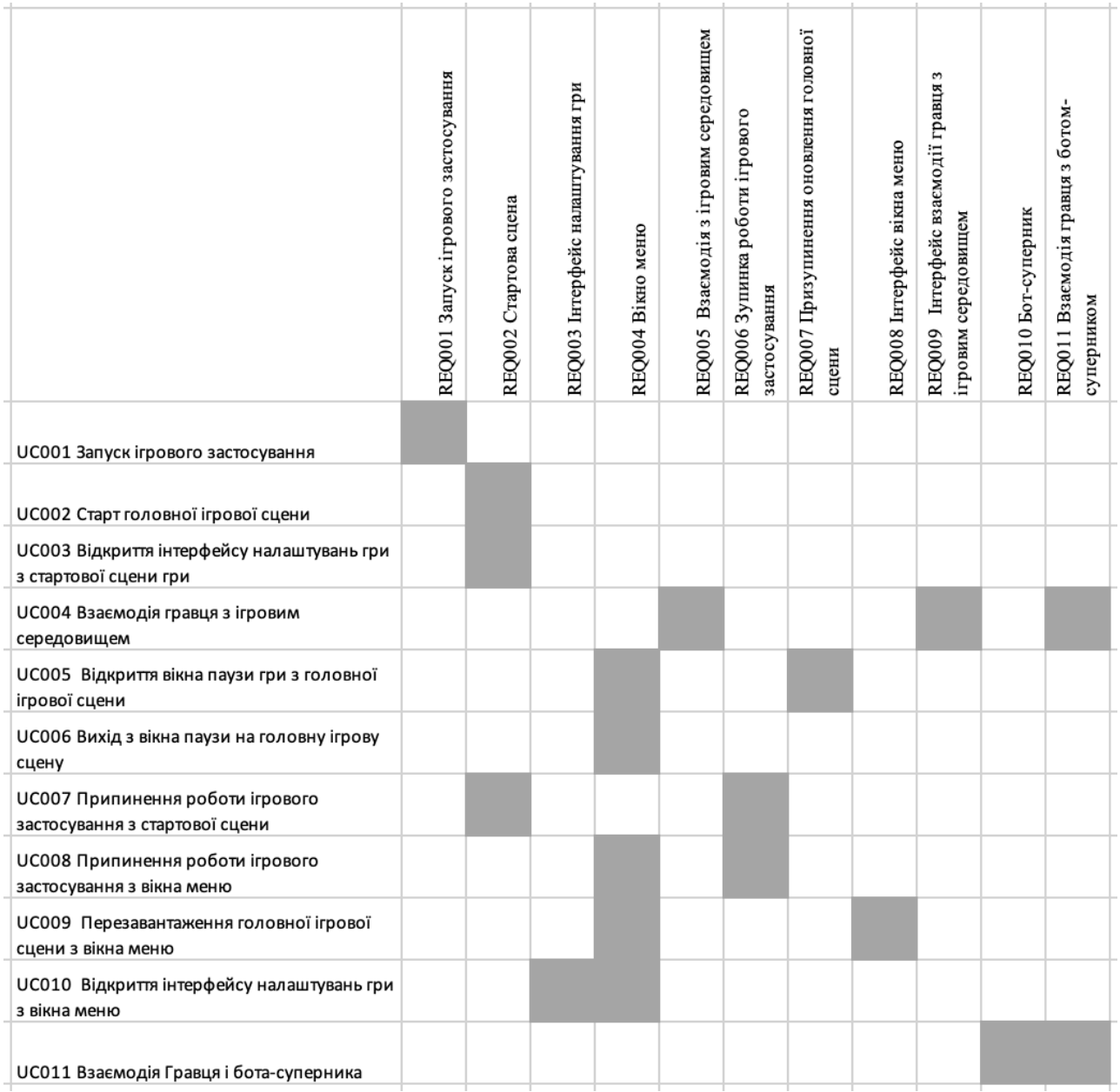


Рисунок 1.1 – Матриця покриття функціональними вимогами варіантів взаємодії

1.4.2 Розроблення нефункціональних вимог

Ігровому застосуванню з навчанням ботів-суперників з використанням нейронних мереж висуваються наступні нефункціональні вимоги:

– має бути виокремлений модуль з моделлю ігрового бота-суперника, яку має бути легко замінити після встановлення Гравцем застосунку. Стан цієї моделі не повинен змінюватись протягом ігрової сесії;

– навчена модель бота-суперника повинна в 90% ігрових партій вигравати суперника, який грає випадковим чином;

– модель бота-суперника має бути навченою на основі нейронної мережі з бібліотеки Tensorflow;

– графічний інтерфейс користувача повинен коректно відображатись на будь-якому пристрої, на який встановлено застосунок;

– керування Гравця з ігровим персонажем повинно відбуватись з частотою оновлення кадрів головної ігрової сцени не менш, ніж 60 кадрів на секунду;

– ігрове застосування повинно бути написано з використанням ігрового рушія Unity та мови програмування C# для простого встановлення на будь-який популярний вид пристроїв, або простої інтеграції його модулів в інше ігрове застосування.

1.4.3 Постановка комплексу завдань модулю

Ігрове застосування з навчанням ботів-суперників з використанням нейронних мереж має на меті продемонструвати використання машинного навчання в галузі ігрової індустрії. Дане рішення покликане підвищити зацікавленість цільової аудиторії до ігрового продукту, що прямо вплине на його розвиток та можливості монетизації.

Для того, щоб досягнути поставленої мети, в ігровому застосуванні має бути реалізовано наступні особливості:

– інтеграція навчання ботів-суперників;

– імплементація архітектурних особливостей, що надаватимуть можливості для інтеграції натренованих моделей в ігрове середовище;

– інтеграція навчених моделей в ігрове середовище;

– коректна взаємодія інтегрованих моделей через ботів-суперників з реальними гравцями.

Ігрове застосування повинне мати якісно реалізовані модулі, що даватиме змогу повторно використовувати їх для інтеграції в схожі ігрові продукти та зрозумілий інтерфейс взаємодії гравця з продуктом.

1.5 Висновки по розділу

В наш час, важливу роль в комфортному середовищі для людського існування, відіграє розвиток комп‘ютерних технологій, який безпосередньо вплинув на всі можливі напрями, галузі, світові тенденції та професії. Сучасне життя не можливо уявити без адаптивних систем, які нас оточують. Це може бути як роботизована лінія на виробництві, так і штучний інтелект, що навчається взаємодіяти з людьми і часто не поступається в адаптивності і якості прийнятих рішень.

Наслідком такого стрімкого розвитку є всебічний інтерес до розробки нових підходів у всіх напрямках галузі інформаційних технологій, однією з яких є ігрова індустрія.

Ігрова індустрія представлена різноплановими ігровими продуктами, які вражають різноманіттям своїх ігрових механік, жанрів та тематик. Саме розвиток цих компонентів ігрових розробок і є рушійною силою для розвитку даної індустрії.

Головною метою розвитку нових підходів до розробки ігрових механік, є підвищення рівня зацікавленості цільової аудиторії до ігрового продукту, що в свою чергу дає можливість для покращення рівня монетизації проекту та прямо впливає на його успіх в цілому. Як зазначалось вище, розвиток IT сфери в цілому, і прогрес у сфері машинного навчання, дає можливість для впровадження нових підходів до розробки і реалізації ігрових механік із використанням досягнень в

сфері штучного інтелекту. Прикладом такого використання є інтеграція машинного навчання в ігрові застосунки, а саме – штучний інтелект на основі нейронної мережі. Успіх даної інновації залежить від правильно побудованої архітектури, що надає можливість для здійснення такої інтеграції. Прикладом такої інтеграції може слугувати навчання ботів-суперників з використанням нейронних мереж. Таке рішення дає змогу створити унікальні схеми поведінки ботів-суперників, що привертатиме увагу потенційних гравців та підтримуватиме зацікавленість вже існуючих до ігрового продукту. Не тривіальність поведінки ігрового бота-суперника може кинути виклик будь-якому досвідченому гравцю, та спонукати його до розвитку своїх ігрових навичок, а також розробки ігрових тактик.

У висновку, ігрове застосування з навчанням ботів-суперників з використанням нейронних мереж суттєво підвищує потенційний успіх ігрового продукту та розширює його користувацьку аудиторію, що в поєднанні з якісним графічним матеріалом, цікавим сюжетом, звуковим супроводом і грамотним менеджментом робить ігровий проект конкурентноспроможним в своєму жанрі та підвищує його шанси на успіх.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для створення належної архітектури ігрового застосування валивим є формалізація варіантів взаємодії з продуктом та функціональні вимоги у вигляді бізнес-процесів, представлених BPMN діаграмами. BPMN діаграма повинна відображати кроки, що відбуваються на кожному етапі бізнес-процесу. Для розроблюваного програмного продукту можна виділити наступні бізнес-процеси:

– взаємодія гравця з користувацьким інтерфейсом;

– взаємодія гравця з ігровим середовищем;

– взаємодія гравця з ботом-суперником.

Схема бізнес-процесу взаємодії гравця з користувацьким інтерфейсом зображена на рисунку 2.1 – Схема бізнес-процесу взаємодії гравця з користувацьким інтерфейсом.

Опис бізнес-процесу взаємодії гравця з користувацьким інтерфейсом:

– гравець натискає на елемент користувацького інтерфейсу через пристрій введення;

– ігрове застосування обробляє сигнал натискання обраного елементу та передає його ідентифікатор до менеджеру користувацького інтерфейсу;

– ідентифікатор обробляється менеджером користувацького інтерфейсу та повертає результат обробки на екран гравця;

– гравець отримує результат взаємодії з елементом користувацького інтерфейсу на своєму екрані.

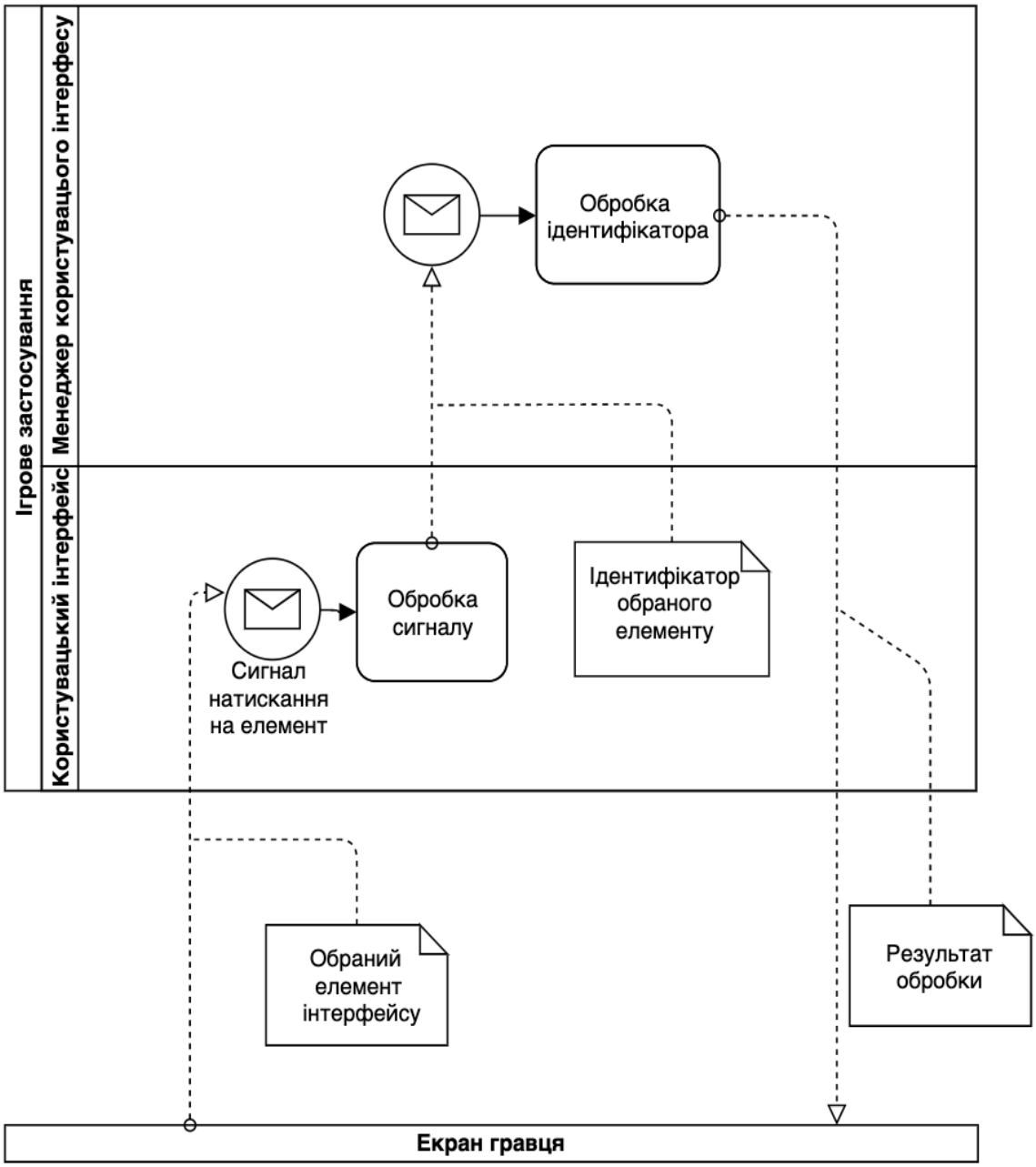


Рисунок 2.1 – Схема бізнес-процесу взаємодії гравця з користувацьким інтерфейсом

Схема бізнес-процесу взаємодії гравця з ігровим середовищем зображена на рисунку 2.2 – Схема бізнес-процесу взаємодії гравця з ігровим середовищем.

Опис бізнес-процесу взаємодії гравця з ігровим середовищем:

– гравець натискає на одну з контролюючих клавіш на пристрої потокового введення;

– ігрове застосування обробляє сигнал натискання клавіші та передає її ідентифікатор на контролер пересування ігрового персонажа гравця;

– контролер здійснює переміщення ігрового персонажа гравця в ігровому середовищі;

– ігрове застосування виводить на екран гравця оновлену позицію його ігрового персонажа.

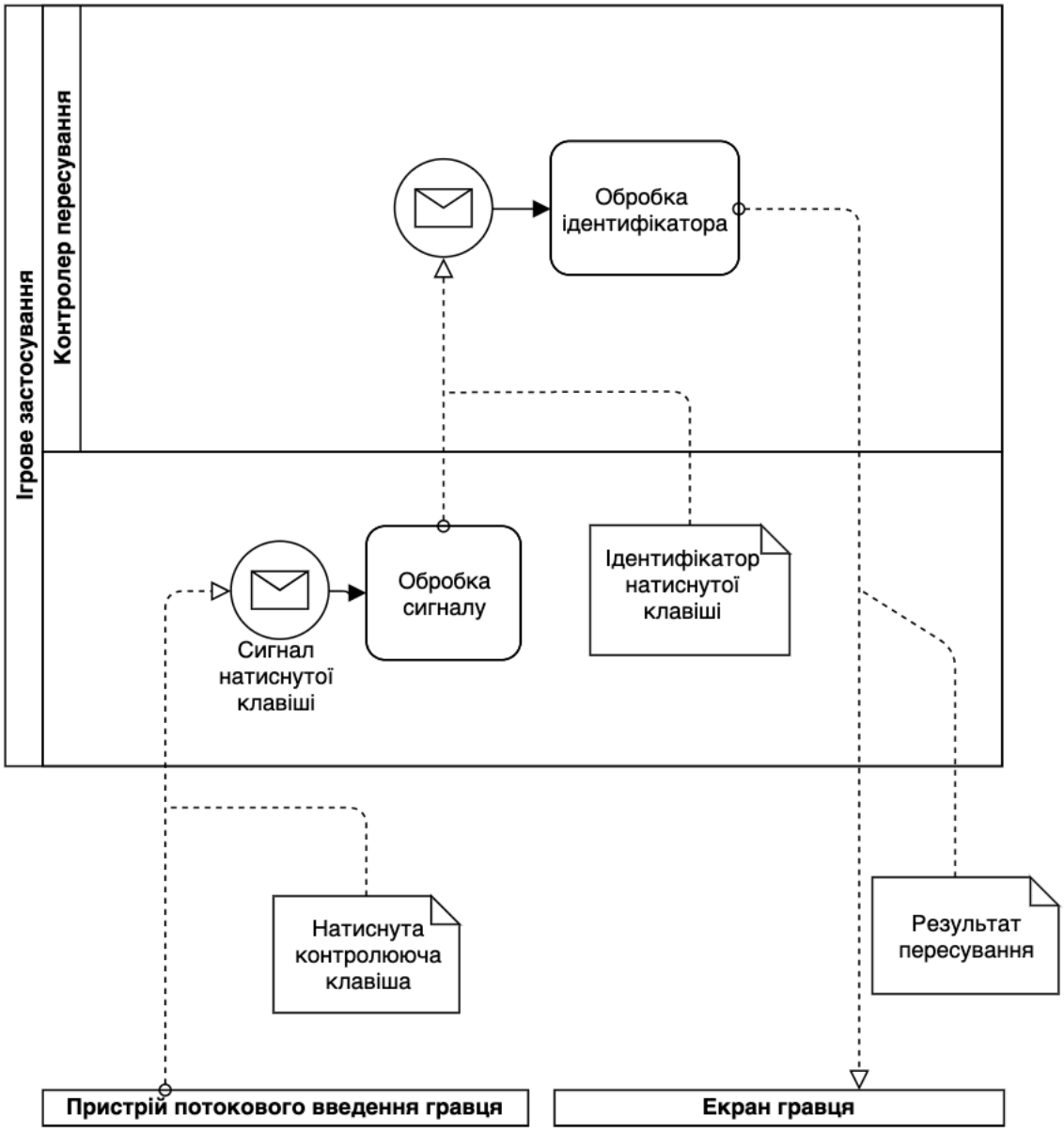


Рисунок 2.2 – Схема бізнес-процесу взаємодії гравця з ігровим середовищем

Схема бізнес процесу взаємодії гравця з ботом-суперником зображена на рисунку 2.3 – Схема бізнес-процесу взаємодії гравця з ботом-суперником.

Опис бізнес-процесу взаємодії гравця з ботом-суперником:

– гравець взаємодіє з ігровим середовищем так, як описано в бізнес-процесі взаємодії гравця з ігровим середовищем;

– ігрове застосування обробляє колізію ігрових персонажів гравця та бота-суперника;

– колізія двох персонажів відображається на екрані гравця та виводиться повідомлення про програш ігрового рівня.

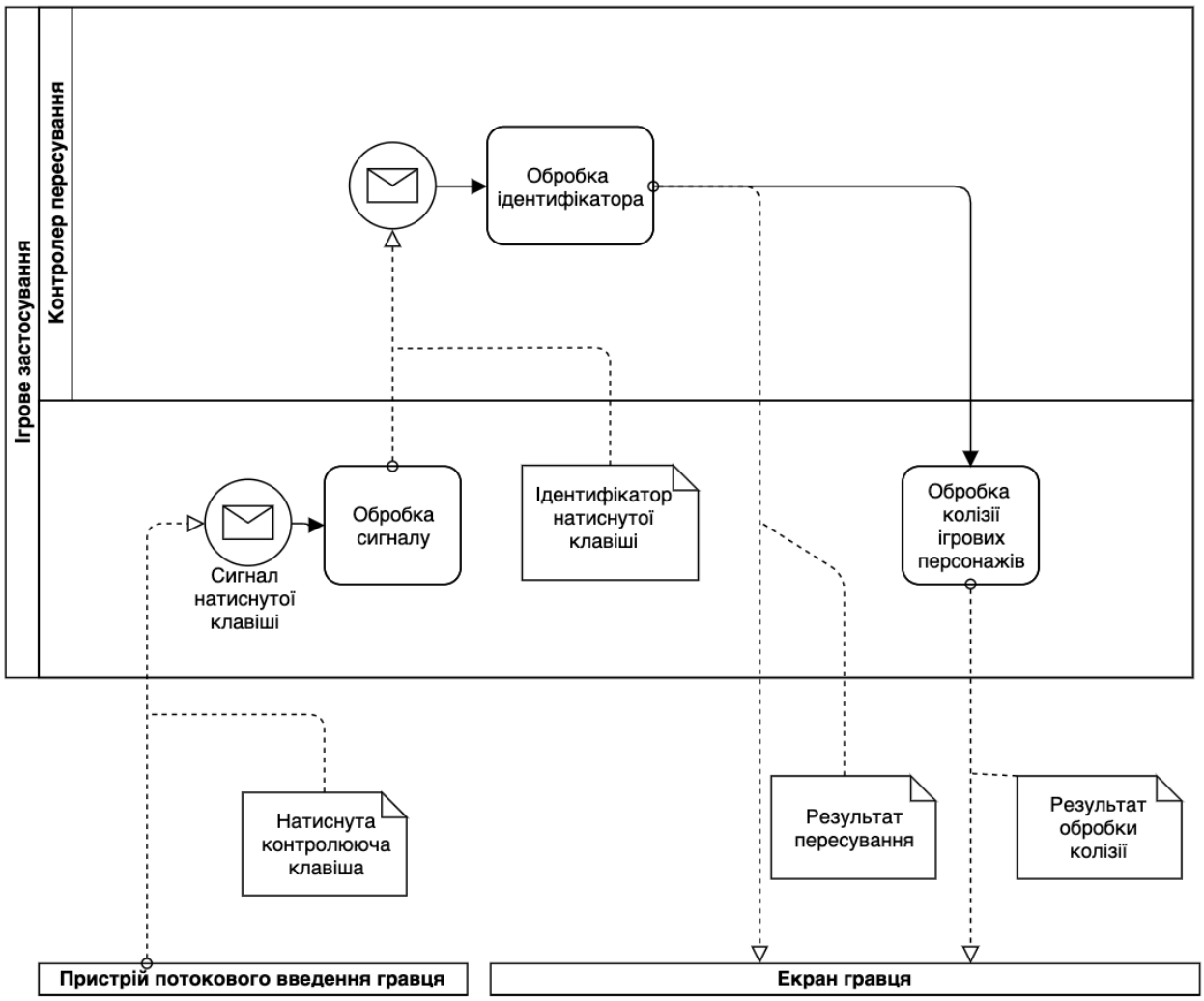


Рисунок 2.3 – Схема бізнес-процесу взаємодії гравця з ботом-суперником

2.2 Архітектура програмного забезпечення

Для забезпечення високої гнучкості та просто масштабування ігрового застосування, його архітектуру було розділено на 5 модулів. Схему розробленої архітектури зображено в документі КПІ.ІП-6312.045490.07.99.СС “Схема структурна компонентів програмного забезпечення”.

Модуль Controller – модуль для інтеграції контролю поведінки об’єктів сцени ігрового застосування. Цей модуль виконує з’єднання об’єктів ігрової сцени з реалізацією сценаріїв їх поведінки на сцені.

Модуль Spawner – модуль для створення, приєднання та надання доступу до параметрів об’єктів на сцені із структур їх логічного представлення. Використовується іншими модулями для створення та приєднання об’єктів до сцени, отримання доступу до властивостей необхідних об’єктів, модифікації та видалення створених об’єктів сцени.

Модуль Maze – модуль, що здійснює всі операції над генерацією структури, що представляє логічну модель лабіринту, генерує сигнал на створення, модифікацію та видалення об’єктного представлення лабіринту на сцені в модуль Spawner та надає доступ і всю необхідну інформацію про стан, параметри та тип лабіринту для всіх інших модулів ігрового застосування.

Модуль Player – модуль призначений для роботи з об’єктними моделями гравців. Модуль здійснює обробку структур логічного представлення гравців в ігровому середовищі, генерує сигнали створення, модифікації та видалення об’єктного представлення гравців в ігровому середовищі в модуль Spawner та надає всю необхідну інформацію про тип, стан і властивості конкретного гравця.

Модуль ML Agents – модуль, що проводить інтеграцію елементів машинного навчання в ігрове середовище, здійснює навчання моделей ботів-суперників та надає інтерфейс для роботи з ними.

Діаграми класів усіх вищеперелічених компонентів зображено в документі КПІ.ІП-6312.045490.07.99.СС “Схема структурна класів програмного забезпечення”.

Для створення сцен та об’єктів на сценах ігрового застосування використано ігровий рушій Unity.

Для задоволення вимог про навчання та інтеграцію ігрових ботів-суперників в розроблюваний продукт, вирішено інтегрувати та використовувати пакет інструментів Unity ML Agents, що дозволяє використовувати нейронні

мережі бібліотеки Tensoflow для навчання і інтеграції моделей в ігрове застосування.

2.3 Конструювання програмного забезпечення

Детальний опис публічних методів класів модулю Controller наведено в наступних таблицях:

Таблиця 2.1 — Публічні методи класу ApplicationContoller

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
StartControllers	-	-	Метод призначений для керованого старту роботи всіх контролерів, що зареєстровано в класі ApplicationController
ResetControllers	-	-	Метод призначений для керованого відновлення стану всіх контролерів, що зареєстровано в класі ApplicationController до початковогостану

Таблиця 2.2 — Публічні методи класу Broadcaster

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
AddSubscription	eventName, listener	-	Метод призначений для створення зв’язку обробника listener з подією eventName

Продовження таблиці 2.2

RemoveSubscription	eventName, listener	-	Метод призначений для видалення зв’язку обробника listener з подією eventName
BroadcastGlobalMessage	eventName	-	Метод призначений для запуску всіх обробників зв’язаних з подією eventName

Таблиця 2.3 — Публічні методи класу UIContoller

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
OnPauseClick	-	-	Метод призначений для оброки натиску на кнопку Pause та запуску обробників сигналу "MainHUD.Hide"
OnResumeClick	-	-	Метод призначений для оброки сигналу натиску на кнопку Resume
OnQuitClick	-	-	Метод призначений для оброки сигналу натиску на кнопку Quit
OnResetClick	-	-	Метод призначений для оброки сигналу натиску на кнопку Reset

Продовження таблиці 2.3

OnSettingsClick	-	-	Метод призначений для оброки сигналу натиску на кнопку Settings

Таблиця 2.4 — Публічні методи класу PlayerFollow

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
getPlayerTransform PlayerTransform	-	Transform	Метод призначений для отримання доступу до компоненту Transform ігрового персонажа. Використовується для реалізації пересування головної камери до позиції ігрового персонажа

Таблиця 2.5 — Публічні методи класу PlayerControlls3D

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
Update	-	-	Метод призначений для асинхронного зчитування і оброки сигналів контролюючих клавіш з пристрою введення гравця та пересування його ігрового персонажа в ігровому середовищі

Таблиця 2.6 — Публічні методи класу RandomMazeWalkController

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
FixedUpdate	-	-	Метод призначений для асинхронного обчислення і реалізації алгоритму випадкового пересування лабіринтом ігрового персонажа

Детальний опис публічних методів класів модулю Spawner наведено в таблицях 2.7, 2.8.

Таблиця 2.7 — Публічні методи класу PlayerSpawner

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
SpawnPlayer	position, playerData	PlayerEntity	Метод призначений для створення об’єкта гравця з характеристиками playerData та його розміщення на сцені в точці position
SpawnPlayers	-	-	Метод призначений для створення всіх об’єктів гравців, що мають знаходитись на сцені
Reset	-	-	Метод призначений для видалення всіх об’єктів гравців з сцени

Таблиця 2.8 — Публічні методи класу PlayerSpawner

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
SpawnMaze	-	-	Метод призначений для створення об’єкта лабіринту і розміщення його на сцені з структури його логічного представлення
Reset	-	-	Метод призначений для видалення об’єкта лабіринту з сцени

Детальний опис публічних методів класів модулю Maze наведено в таблицях 2.9, 2.10, 2.11.

Таблиця 2.9 — Публічні методи класу Maze

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
GetCellScreenPosition	cell	Vector3	Метод призначений для обчислення позиції центру комірки лабіринту cell
GetRandomCell	-	MazeGeneratorCell	Метод призначений для отримання випадкової комірки лабіринту

Продовження таблиці 2.9

GetRandomFreeCell	-	MazeGeneratorCell	Метод призначений для отримання випадкової комірки лабіринту, на якій не знаходиться жодний гравець
GetAccesibleCells	cell	List <MazeGeneratorCell>	Метод призначений для пошуку та надання доступу до доступних для пересування в них комірок від комірки лабіринту cell
GetIndexPosition	x, y, ignoringSize	MazeGeneratorCell	Метод призначений для отримання позиціх комірки з врахуванням її розмірів, або без урахування по індексу [x, y]
GetClosestCell	position	MazeGeneratorCell	Метод призначений для отримання пошуку найближчої комірки лабіринту до точки position

Продовження таблиці 2.9

NormalizedPosition	position	Vector3	Метод призначений для отримання нормалізованої позиції position відносно розмірів лабіринту
Reset	-	-	Метод призначений для очистки всіх даних про логічну представлення лабіринту
DistanceBetweenCells	start, target	int	Метод призначений для пошуку найкоротшого шляху між комірками start та target хвильовим алгоритмом Лі. Якщо такого шляху не існує, алгоритм поверне від’ємне значення

Таблиця 2.10 — Публічні методи класу MazeGenerator

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
GenerateMaze	currentMaze	MazeGeneratorCell [,]	Метод призначений для генерації двомірного масиву логічного представлення лабіринту методом зворотнього трасування з параметрами сутності currentMaze

Таблиця 2.11 — Публічні методи класу MazeManager

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
GetMazeEntity	-	Maze	Метод призначений для генерації для отримання сутності лабіринту
Reset	-	-	Метод призначений для скидання параметрів об’єкту класу MazeManager

Детальний опис публічних методів класів модулю Player наведено в таблицях 2.12, 2.13.

Таблиця 2.12 — Публічні методи класу PlayerManager

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
GetPlayers	-	List<PlayerEntity>	Метод призначений для отримання доступу до списку сутностей гравців

Продовження таблиці 2.12

RemovePlayer	player	bool	Метод призначений для видалення гравця player з списку гравців
Reset	-	-	Метод призначений для скидання параметрів об’єкту класу PlayerManager

Таблиця 2.13 — Публічні методи класу PlayerData

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
getPrefab prefab	-	PlayerEntity	Метод призначений для отримання доступу до ігрового персонажа гравця, що буде створено і додано на ігрову сцену для взаємодії з ігровим середовищем
getSpeed Speed	-	float	Метод призначений для отримання значення швидкості пересування ігрового персонажа гравця і використовується його відповідним контролером

Продовження таблиці 2.13

getType Type	-	PlayerType	Метод призначений для отримання типу ігрового персонажа гравця і використовується контролером обрахунку колізій ігрових персонажів

Детальний опис публічних методів класів модулю ML Agents наведено в таблицях 2.14, 2.15.

Таблиця 2.14 — Публічні методи класу PersecutorAgent

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
Initialize	-	-	Метод призначений для ініціалізації необхідних змінних при кожному початку епізоду навчання, або використання моделі
OnEpisodeBegin	-	-	Метод призначений для визначення стану агента на початку епізоду навчання, або використання моделі
CollectObservations	sensor	-	Метод призначений для зберігання у вектор sensor даних, що є вхідними для нейронної мережі та збираються на кожній ітерації навчання або використання моделі

Продовження таблиці 2.14

MoveAgent	actions	-	Метод призначений для інтерпретації вихідного шару нейронної мережі, представленого у вигляді масиву actions типу float в пересування ігрового персонажа агента в ігровому середовищі
OnActionReceived	vectorAction	-	Метод призначений для зчитування вихідного шару нейронної мережі у масив vectorAction, запуску пересування ігрового персонажа бота-суперника та винагородження агента за досягнення чи недосягнення поставлених цілей
Heuristic	actionsOut	-	Метод призначений для реалізації ручного керування через пристрій потокового введення при навчанні моделі нейронної мережі із записом виконаних дій в масив actionsOut. При цьому модель продовжує навчатись і використовує actionsOut як вихідний шар нейронної мережі

Продовження таблиці 2.14

RegisterLose	-	-	Метод призначений для дострокового завершення епізоду навчання або використання моделі з реєстрацією негативного значення винагороди
EndEpisodeAndReset	-	-	Метод призначений для дострокового завершення епізоду навчання або використання моделі та запуску процесу глобального скидання стану ігрового середовища

Таблиця 2.15 — Публічні методи класу Agent

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
LazyInitialize	-	-	Метод призначений для ініціалізації безпечної ініціалізації агента до об’єкту Academy, що дає змогу одночасно тренувати декілька агентів на одній сцені
SetModel	behaviorName, model, inferenceDevice	-	Метод призначений для внутрішньої ініціалізації агнету моделлю типу behaviorName об'єктом model з інтерфейсом тренування inferenceDevice

Продовженні таблиці 2.15

StepCount	-	int	Метод призначений для отримання значення кількості виконаних кроків в епізоді
SetReward	reward	-	Метод призначений для реєстрації в розпочатому епізоді нагороди агента із значенням reward
AddReward	reward	-	Метод призначений для інкрементної зміни значення кумулятивної нагороди агента на значення reward в розпочатому епізоді
EndEpisode	-	-	Метод призначений для закінчення розпочатого епізоду
RequestDecision	-	-	Метод призначений для створення запиту на прийняття рішення моделлю нейронної мережі, що знаходиться в стані навчання та прив’язана до агента

Перелік з детальним описом сцен ігрового застосування наведено в таблиці 2.16.

Таблиця 2.16 – Опис сцен ігрового застосування

Назва сцени	Перелік об’єктів сцени	Призначення сцени
Maze3D	Main Camera, Broadcaster, ApplicationController, ScriptableObjectsController, MazeSpawner, PlayerSpawner, UI, Hint, Directional Light, EventSystem	Сцена призначена для розміщення на ній ігрового середовища, ігрових персонажів гравців, об’єктів класів модуля Controller, Spawner, Maze, Player та ML Agents. На даній сцені відбувається відображення взаємодії гравця та ігрового середовища, гравця з інтерфейсом користувача та гравця з ботом-суперником
Start	Main Camera, AgentCube_Purple, AgentCube_Blue, Floor, Sky, Clouds, Directional Light, Canvas, EventSystem	Сцена призначена для розміщення на ній стартового екрану ігрового застосування та відображення на ній інтерфейсу користувача

Перелік об’єктів, їх властивості та компоненти можуть змінюватись в залежності від стану, в якому перебуває ігрове застосування.

На рисунках 2.4, 2.5 зображено ієрархію об’єктів сцен Maze3D та Start відповідно.

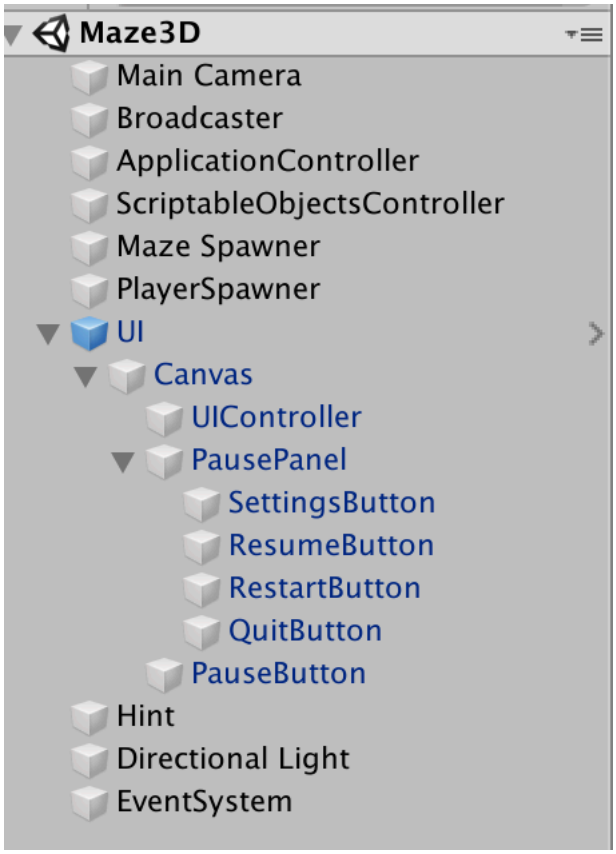


Рисунок 2.4 – Ієрархія основних об’єктів сцени Maze3D

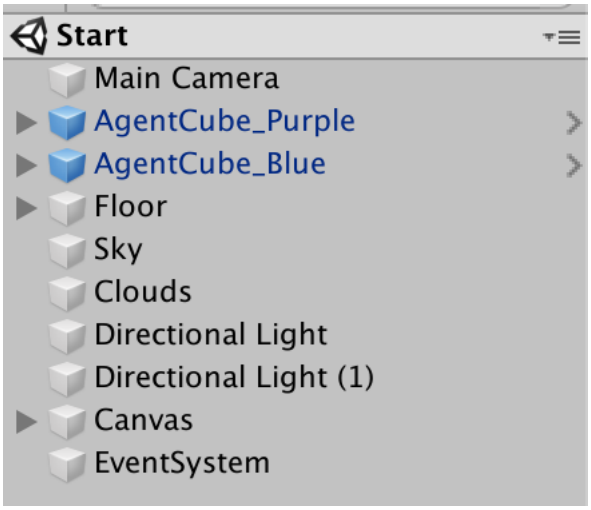


Рисунок 2.5 – Ієрархія основних об’єктів сцени Start

2.4 Тренування ботів-суперників

Процес інтеграції машинного навчання ботів-суперників в ігрове застосування передбачає виконання наступних кроків:

– інтеграції пакету інструментів розробника Unity ML Agents в середовище розробки Unity;

– встановлення та налаштування необхідного для роботи пакету програмного середовища, що включає в себе ряд бібліотек та застосунків;

– створення ігрового середовища, що надає інтерфейс для скидання його стану до початкового, після переходу до наступного епізоду навчання моделі інтегрованого в нього агента;

– налаштування агента та його подальша інтеграція в ігрове середовище;

– розробка архітектури нейронної мережі;

– розробка плану ітераційного навчання з урахуванням успішності прийнятних агентом рішень;

– тренування та тестування результатів навчання моделі.

2.4.1 Інтеграція пакету інструментів розробника Unity ML Agents в середовище розробки Unity

Інтеграція пакету інструментів починається з завантаження програмних кодів пакету Unity ML Agents. Зробити це можна двома способами:

– з розділу Releases & Documentation їх репозиторія на GitHub;

– виконавши команду клонування віддаленого репозиторія GitHub.

Команду клонування віддаленого репозиторія GitHub представлено на рисунку 2.6 – Команда клонування віддаленого репозиторія GitHub пакету інструментів розробника Unity ML Agents.

```
git clone --branch release_1 https://github.com/Unity-Technologies/ml-agents.git
```

Рисунок 2.6 – Команда клонування віддаленого репозиторія GitHub пакету інструментів розробника Unity ML Agents

Після завантаження, необхідно інтегрувати пакет до проекту Unity. Необхідні кроки для інтеграції зображено на рисунку 2.7 – Інтеграція пакету розробника Unity ML Agents до проекту Unity

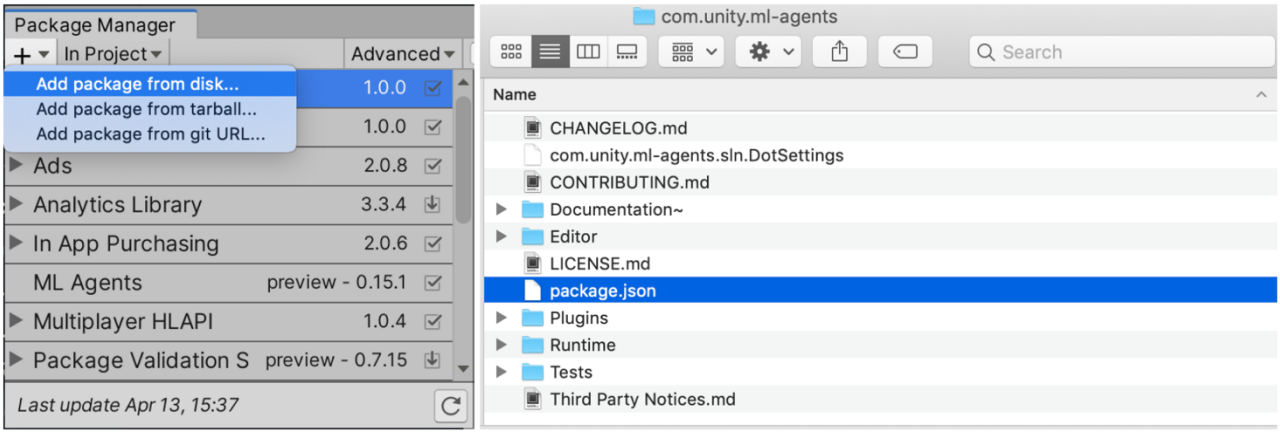


Рисунок 2.7 – Інтеграція пакету розробника Unity ML Agents до проекту Unity

2.4.2 Налаштування необхідного програмного середовища

Налаштування програмного середовища відбувається шляхом виконання наступних команд командного рядка зображених на рисунку 2.8 – Команди налаштування програмного середовища.

```
pip3 install mlagents
pip3 install -e ./ml-agents-envs
pip3 install -e ./ml-agents
```

Рисунок 2.8 – Команди налаштування програмного середовища

2.4.3 Налаштування агента та його інтеграція в ігрове середовище

Налаштування агента відбувається в редакторі Unity. Налаштування параметрів агента – важливий аспект, якому необхідно приділити достатньо уваги. Вибір неправильних налаштувань може призвести до непередбачуваної поведінки агента та суттєво ускладнити процес тренування моделі. Пояснення з призначення компонентів та їх параметрів описано в документації до пакету розробника Unity ML Agents. Параметри налаштувань агента зображено на рисунку 2.9 – Налаштування агента

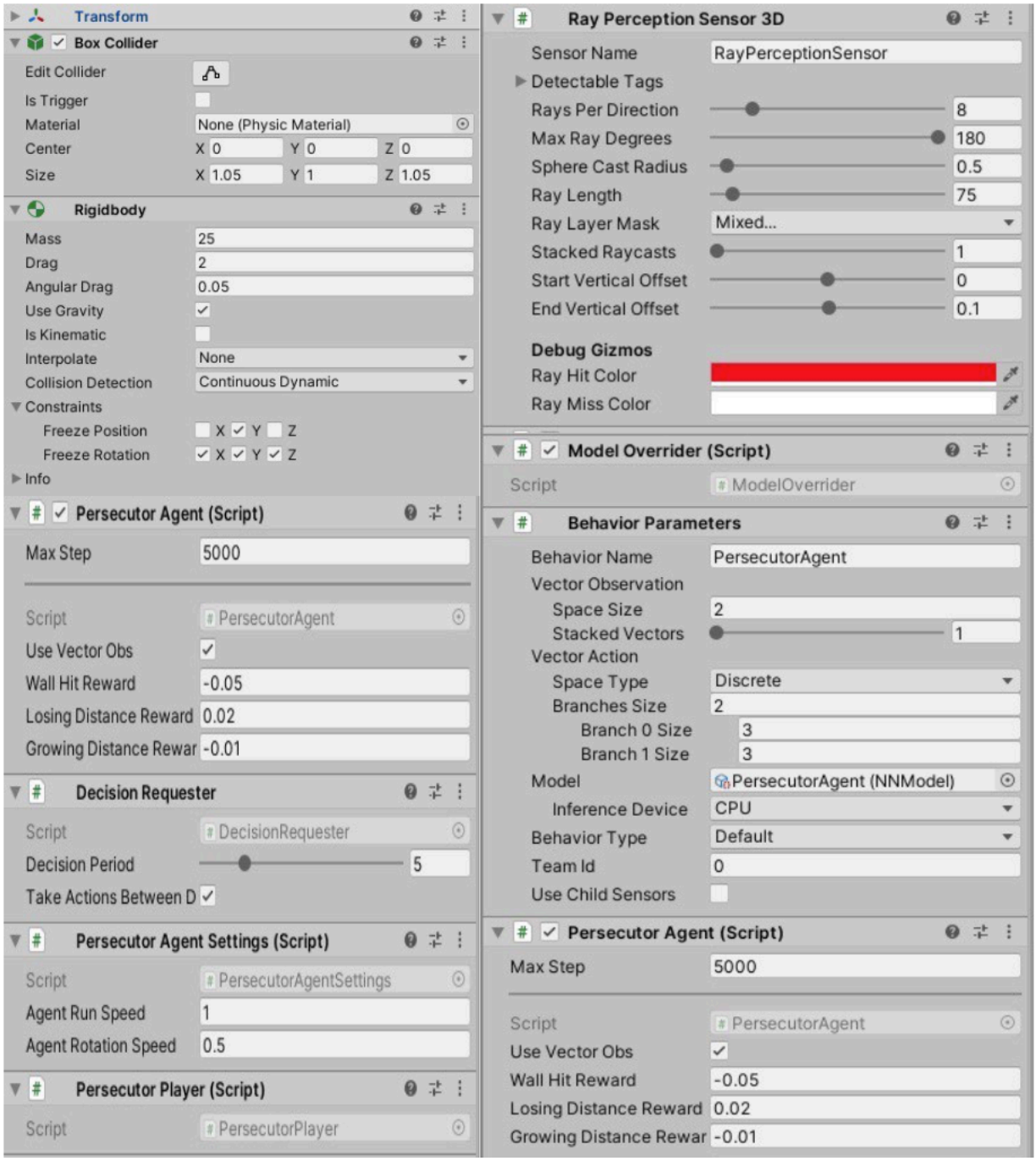


Рисунок 2.9 – Налаштування агента

2.4.4 Розробка архітектури нейронної мережі

Розробка архітектури моделі нейронної мережі полягає в створенні файлу налаштувань, який буде зчитуватись при кожному запуску тренування та визначатиме головні властивості мережі, розмір буферу навчання, кількість кроків, тощо. Параметри розробленої архітектури зображено в рисунку 2.10 – Налаштування архітектури нейронної мережі.

```
PersecutorAgent:
    use_recurrent: true
    sequence_length: 64
    num_layers: 2
    hidden_units: 128
    memory_size: 128
    beta: 1.0e-2
    num_epoch: 3
    buffer_size: 1024
    batch_size: 128
    max_steps: 2.5e7
    summary_freq: 10000
    time_horizon: 64
```

Рисунок 2.10 – Налаштування архітектури нейронної мережі

2.4.5 Розробка плану ітераційного навчання

План ітераційного навчання описується у вигляді .yaml файлу та передається до команди запуску тренування як один з параметрів. Розроблений план навчання зображено на рисунку 2.11 – Ітераційний план навчання моделі нейронної мережі.

```
PersecutorAgent:
  measure: reward
  thresholds: [-0.5, -0.1, 0.3, 0.5, 0.8, 1.2, 1.5]
  min_lesson_length: 80
  signal_smoothing: true
  parameters:
    escaper_speed: [0.2, 0.25, 0.25, 0.30, 0.35, 0.40, 0.50, 1.0]
    catching_distance: [3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 2.0, 1.0]
    maze: [1.0, 1.0, 2.0, 2.0, 3.0, 3.0, 4.0, 4.0]
```

Рисунок 2.11 – Ітераційний план навчання моделі нейронної мережі

2.4.6 Тренування та тестування результатів навчання моделі

По закінченню налаштування параметрів агента та нейронної мережі, можна приступити до початку навчання моделі. Навчання починається із запуску відповідної команди утиліти командного рядка *mlagents-learn* з вказанням конфігураційних файлів моделі нейронної мережі та плану ітераційного навчання

та стару ігрової сцени Unity. Проміжні результати навчання відображуються в командному рядку у вигляді відповідних повідомлень про кумулятивну нагороду агента, кількість зроблених кроків та час, протягом якого відбувається навчання. Приклад таких повідомлень зображено на рисунку 2.12 – Проміжні результати навчання моделі.

```
2020-05-14 22:34:13 INFO [stats.py:111] persecutor_base_1_01_PersecutorAgent: Step: 17320000
. Time Elapsed: 30347.567 s Mean Reward: -0.011. Std of Reward: 1.935. Training.
2020-05-14 22:35:02 INFO [stats.py:111] persecutor_base_1_01_PersecutorAgent: Step: 17330000
. Time Elapsed: 30396.374 s Mean Reward: 0.068. Std of Reward: 2.424. Training.
2020-05-14 22:35:50 INFO [stats.py:111] persecutor_base_1_01_PersecutorAgent: Step: 17340000
. Time Elapsed: 30444.096 s Mean Reward: -0.844. Std of Reward: 4.102. Training.
2020-05-14 22:36:38 INFO [stats.py:111] persecutor_base_1_01_PersecutorAgent: Step: 17350000
. Time Elapsed: 30492.556 s Mean Reward: 0.238. Std of Reward: 1.527. Training.
2020-05-14 22:37:26 INFO [stats.py:111] persecutor_base_1_01_PersecutorAgent: Step: 17360000
. Time Elapsed: 30541.005 s Mean Reward: -0.331. Std of Reward: 2.976. Training.
2020-05-14 22:38:07 INFO [trainer_controller.py:112] Saved Model
```

Рисунок 2.12 – Проміжні результати навчання моделі

По закінченню тренування моделі, можна оцінити якість побудованої архітектури моделі та плану ітераційного тренування. Також, важливим є питання, чи було тренування моделі рентабельним. Оцінивши графіки статистики тренувань бота-суперника, можна з впевненістю сказати, що навчена модель є достатньо ефективною, що свідчить на користь використання такого підходу в ігрових розробках.

Для візуалізації зібраної статистики використаємо утиліту Tensorboard, яка надає можливість представити зібрану статистику у вигляді графіків.

Графік зміни отриманої агентом нагороди зображено на рисунку 2.13 – Графік залежності отриманої винагороди від кількості кроків навчання.

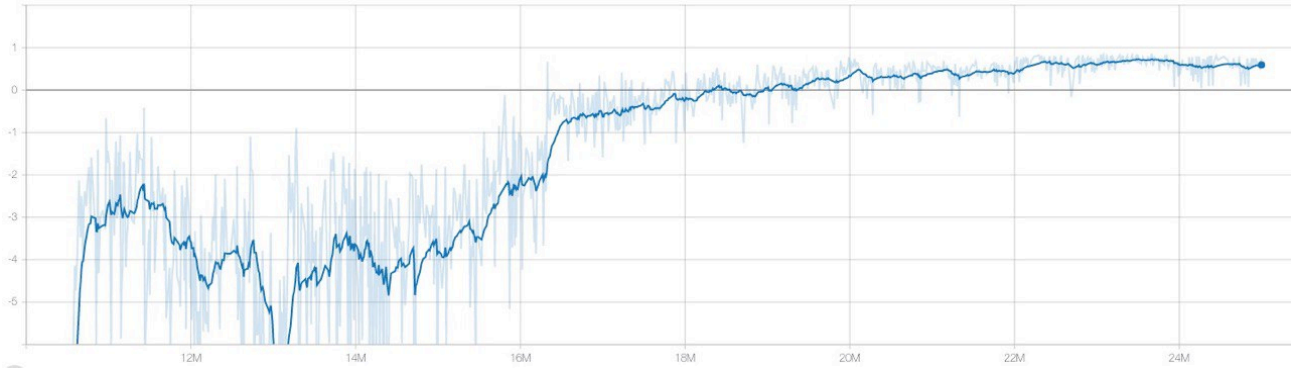


Рисунок 2.13 – Графік залежності отриманої винагороди від кількості кроків навчання

Як видно з графіку, нагорода, яку отримував агент за досягнення поставлених цілей, з ростом кількості навчальних кроків, зростає. Це прямо свідчить про підвищення якості прийнятих рішень в розрізі поставлених агенту задач.

Графік зміни розміру навчального епізоду представлено на рисунку 2.14 – Графік залежності розміру епізоду від кількості кроків навчання.

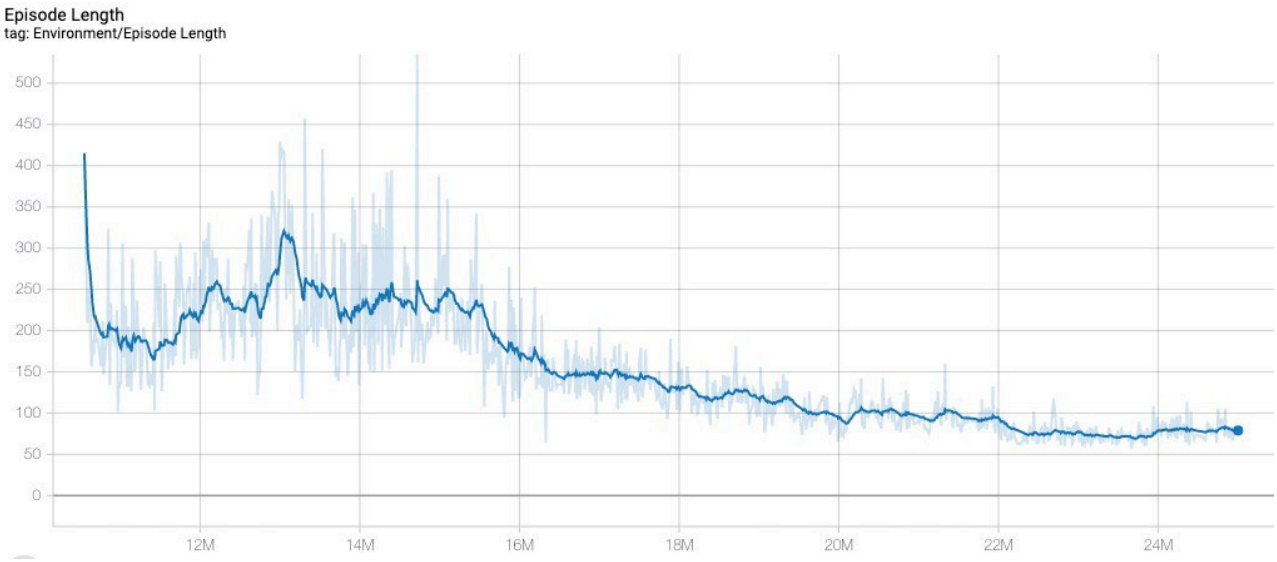


Рисунок 2.14 – Графік залежності розміру епізоду від кількості кроків навчання

Як видно з наведеного графіку, довжина тренувального епізоду зменшується по мірі навчання моделі. Це означає, що модель в ході тренувань навчилась все швидше і швидше досягати поставлених цілей, що робить її ефективнішою.

Графік зміни етапів ітераційного навчання представлено на рисунку 2.15 – Графік залежності зміни етапу ітераційного навчання від кількості кроків навчання.

Оцінивши графік, можна впевнено свідчити про якість розробленого плану навчання моделі. Модель не застрягала на певних рівнях навчання, та впевнено прогресувала в швидкості і якості прийнятті рішень в умовах ускладнення поставлених задач.

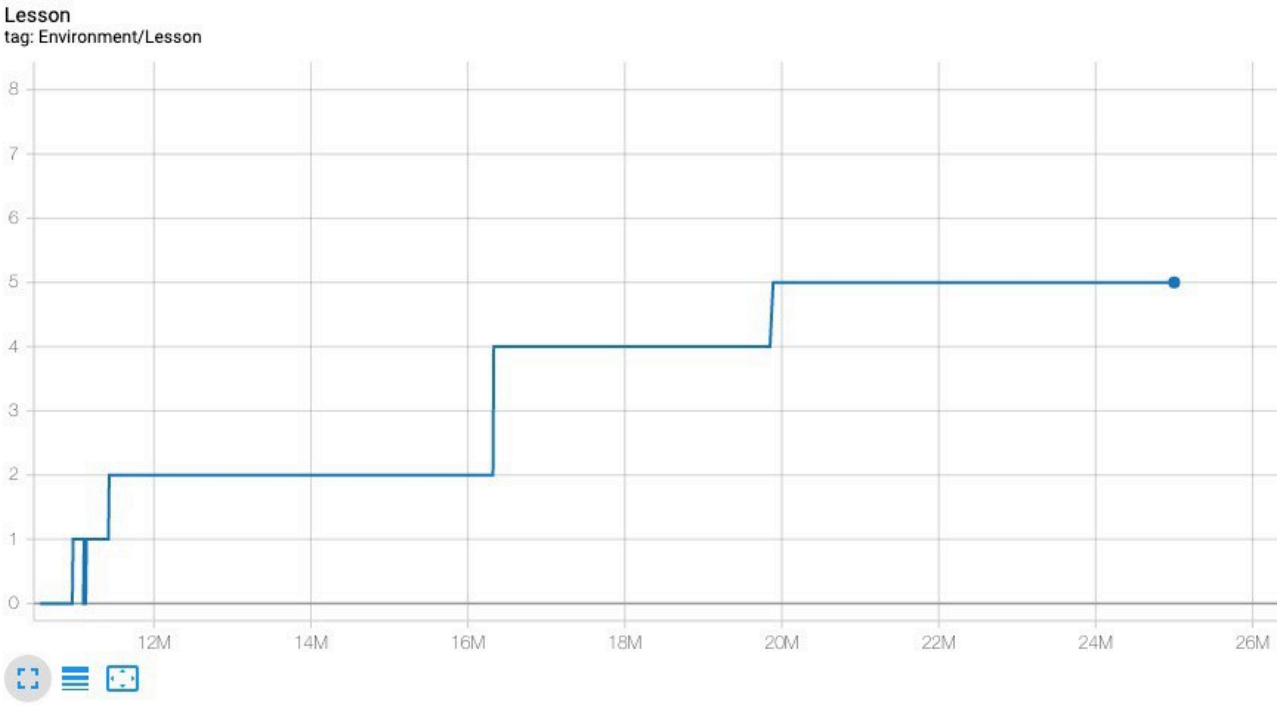


Рисунок 2.15 – Графік залежності зміни етапу ітераційного навчання від кількості кроків навчання

2.5 Аналіз безпеки даних

Розроблений програмний продукт не передбачає зберігання чи передачу даних через будь-яку мережу. Ігрове застосування не збирає і не зберігає будь-яких вразливих до атаки чи персональних даних. Відповідно, в інтеграції алгоритмів шифрування, використанні хеш-функцій або захищений протоколів передачі даних немає необхідності.

2.6 Висновки по розділу

В даному розділі було проведено аналіз та виокремлення основних бізнес-процесів для ігрового застосування з навчанням ботів-суперників з використанням нейронних мереж, враховуючи можливі варіанти взаємодії та сценарії використання. Виділені бізнес-процеси було зображено у вигляді BPMN діаграм.

Також було створено архітектуру розроблюваного програмного забезпечення та проведено детальний опис її модулів, описано основні методи

розроблених класів. Архітектуру ігрового застосування зображено у вигляді діаграм компонентів та класів.

Детально описано процес навчання ботів-суперників, наведено приклад розробленого агента, описано основні підходи до побудови плану ітераційного навчання та наведено статистичні дані, які було проаналізовано та зроблено висновки про успішність побудованого процесу навчання та підходу в цілому.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Якість програмного забезпечення – це набір таких характеристик, які мають відповідати встановленим та очікуваним вимогам до програмного продукту. Оцінити відповідність розроблюваного програмного забезпечення встановленим вимогам можна з допомогою тестування. Отже, тестування є невід'ємною складовою процесу розробки кожного програмного забезпечення.

Основними складовими якості програмного забезпечення є:

– надійність – спроможність ПЗ виконувати задачі детерміновано – за певний проміжок часу, або за відведену кількість операцій;

– функціональність – здатність ПЗ виконувати дії, що покликані вирішити задачі відповідно до потреб користувача;

– зручність використання – властивість ПЗ, що дає можливість користувачу легкого розуміти принципи використання продукту та бути привабливим для нього;

– ефективність – спроможність ПЗ до забезпечення достатнього рівня продуктивності;

– портативність – характеристика ПЗ, що показує легкість його перенесення для використання в інших програмних середовищах.

Метою даного розділу є описання процесу тестування ігрового застосування з навчання ботів-суперників з використанням нейронних мереж.

3.2 Опис процесів тестування

Для того, щоб оцінити якість ігрового застосування з навчанням ботів-суперників з використанням нейронних мереж потрібно оцінити якість наступних об'єктів:

– користувацького інтерфейсу;

– ігрового середовища;

– натренованої моделі бота-суперника.

Для цього потрібно протестувати наступні компоненти:

– взаємодія з елементами користувацького інтерфейсу стартової сцени;

– взаємодія з елементами користувацького інтерфейсу головної ігрової сцени;

– взаємодія з елементами користувацького інтерфейсу вікна паузи;

– взаємодія ігрових персонажів з елементами ігрового середовища;

– взаємодія з ігровим персонажем бота-суперника;

– якість прийняття рішень натренованою моделлю нейронної мережі в контексті протидії гравцю;

– якість прийняття рішень натренованою моделлю нейронної мережі в контексті взаємодії з ігровим середовищем;

– наявність неочікуваних сценаріїв поведінки натренованої моделі нейронної мережі;

– коректність призупинення і продовження ігрової сесії при відкриття вікна паузи та згортанні ігрового застосування;

– генерація лабіринту з циклами;

– генерація лабіринту без циклів;

– пошук найкоротшого шляху в лабіринті між двома комірками.

Так як тестування будуть проводитись розробником, то їх доцільно провести у режимі білої скриньки. Для забезпечення вищого рівня надійності функціонал вирішено протестувати як при негативних умовах, так і при позитивних.

Отже, функціональні тести, що необхідно провести:

– тестування взаємодії ігрових персонажів з елементами ігрового середовища;

– тестування взаємодії з ігровим персонажем бота-суперника;

– тестування на коректність поведінки натренованої моделі нейронної мережі;

– тестування роботи алгоритму пошуку найкоротшого шляху між двома комірками лабіринту;

– тестування алгоритму генерації лабіринту без циклів;

– тестування алгоритму генерації лабіринту з циклами;

До нефункціональних тестів належать:

– тестування користувацького інтерфейсу стартової сцени, головної ігрової сцени та вікна паузи;

– тестування якості результатів прийнятих рішень натренованої моделі нейронної мережі в контексті взаємодії з ігровим середовищем та протидії гравцю.

3.3 Опис контрольного прикладу

В ході тестувань було перевірено функціональність усіх модулів ігрового застосування. Послідовно було протестовано всі варіанти взаємодії та сценарії використання ігрового застосування.

Опис контрольних прикладів наведений у таблицях 3.1, 3.2, 3.3.

Таблиця 3.1 – Перевірка алгоритму генерації лабіринту без циклів

Мета тесту	Перевірка алгоритму генерації лабіринту без циклів
Передумови	Ігровий рівень передбачає використання лабіринту без циклів, ігрове середовище знаходиться в активному стані
Схема виконання	Запустити початок ігрового рівня
Очікуваний результат	Згенерований лабіринт матиме лише один вихід, між двома довільними комірками лабіринту буде існувати лише один шлях
Дійсний результат	Згенерований лабіринт має лише один вихід та задовольняє вимогу до існування лише одного шляху між двома довільними комірками.

Таблиця 3.2 – Перевірка взаємодії ігрових персонажів з елементами ігрового середовища

Мета тесту	Перевірка взаємодії ігрових персонажів з елементами ігрового середовища
Передумови	Ігрове середовище знаходиться в активному стані, існує згенерований лабіринт та в ньому розміщено ігрового персонажа, що контролюється гравцем
Схема виконання	Контролюючими клавішами потокового пристрою введення здійснити пересування ігрового персонажа до найближчої стінки або елемента лабіринту
Очікуваний результат	Ігровий персонаж зіштовхується з елементом ігрового середовища та поводить себе як абсолютно тверде тіло при зіткненні з іншим абсолютно твердим тілом
Дійсний результат	Ігровий персонаж зіштовхується з елементом ігрового середовища та поводить себе як абсолютно тверде тіло при зіткненні з іншим абсолютно твердим тілом

Таблиця 3.3 – Перевірка користувацького інтерфейсу головної ігрової сцени

Мета тесту	Перевірка користувацького інтерфейсу головної ігрової сцени
Передумови	На екрані ігрового застосування активна головна ігрова сцена, та ігрове середовище знаходиться в активному стані
Схема виконання	Оптичним пристроєм введення натисну на кнопку паузи в правому верхньому кутку екрану ігрового застосування
Очікуваний результат	Відкриття вікна паузи, переведення ігрового середовища в стан паузи
Дійсний результат	Відкрито вікно паузи, ігрове середовище переведено в стан паузи

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Ігрове застосування з навчанням ботів-суперників з використанням нейронних мереж розроблено за допомогою ігрового рушія Unity, що підтримує всі популярні платформи та надає вбудований механізм для збірки виконуваних файлів під Windows, Mac OS, IOS, Android та інші.

Для розгортання програмного забезпечення необхідно встановити відповідний наявний варіант збірки на свій пристрій.

4.2 Робота з програмним забезпеченням

Детальний опис роботи з ігровим застосування з навчанням ботів-суперників з використанням нейронних мереж у документі КПІ.ІП-6312.045490.06.34 “Керівництво користувача”.

ВИСНОВКИ

В даному дипломному проекті проаналізовано підходи до інтеграції нейронних мереж в ігрові застосування. Розроблене ігрове застосування з навчанням ботів-суперників з використанням нейронних мереж має на меті провести демонстрацію можливостей використання машинного навчання в галузі ігрової індустрії.

Спроектовано і розроблено ігрове застосування з використанням ігрового рушія Unity та проведено інтеграцію навчання ботів-суперників на основі нейронних мереж.

Досягнення поставленої мети виявилось можливим завдяки реалізації наступних особливостей:

– інтеграції навчання ботів-суперників;

– імплементації архітектурних особливостей, що надаватимуть можливості для інтеграції натренованих моделей в ігрове середовище;

– інтеграції навчених моделей в ігрове середовище;

– високій якості натренованості ботів-суперників.

Готовий програмний продукт пройшов всі етапи тестувань. Виявлені під час тестування помилки було проаналізовано, доопрацьовано та виправлено.

Розроблено проектну документацію, структурні схеми варіантів використання, компонентів та класів програмного забезпечення, а також керівництво користувача.

ПЕРЕЛІК ПОСИЛАНЬ

1) Introduction to TensorFlow [Електронний ресурс]– Режим доступу до ресурсу: https://www.tensorflow.org/learn.

2) Unity ML-Agents Toolkit Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Unity-Technologies/ml-agents/tree/master/docs.

3) Unity User Manual [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/index.html.

4) Comi M. How to teach AI to play Games: Deep Reinforcement Learning [Електронний ресурс] / Mauro Comi – Режим доступу до ресурсу: https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a.

5) Integrating Machine Learning into Game Development [Електронний ресурс] – Режим доступу до ресурсу: https://hackernoon.com/integrating-machine-learning-into-game-development-c5a7f31ed839.

6) Designing a Learning Environment [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Design.md.

7) Training Configuration File [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Configuration-File.md

8) Nicholson C. A Beginner's Guide to Deep Reinforcement Learning [Електронний ресурс] / Chris Nicholson – Режим доступу до ресурсу: https://pathmind.com/wiki/deep-reinforcement-learning.

9) Sutton R. Reinforcement Learning: An Introduction / R. Sutton, A. Barto. – Лондон: The MIT Press Cambridge, 2017. – 445 с. – (5).

10) Simonini T. An Introduction to Unity ML-Agents [Електронний ресурс] / Thomas Simonini – Режим доступу до ресурсу: https://towardsdatascience.com/an-introduction-to-unity-ml-agents-6238452fcf4c.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ІГРОВЕ ЗАСТУВАННЯ З НАВЧАННЯМ БОТІВ-СУПЕРНИКІВ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Технічне завдання

КПІ.ІП-6312.045490.03.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.В

Ковтунець

Виконавець:

_____А.В. Зозуля

Нормоконтроль:

_____ К.І. Ліщук

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Ігрове застосування з навчанням ботів-суперників з використанням нейронних мереж

Галузь застосування:

Наведене технічне завдання поширюється на розробку ігрового застосування з навчанням ботів-суперників з використанням нейронних мереж, котре використовується тренування та використання ботів-суперників та призначене для будь-кого.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки ігрового застосування з навчанням ботів-суперників з використанням нейронних мереж є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації і управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» («КПІ ім. Ігоря Сікорського»).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для демонстрації можливостей інтеграції машинного навчання в ігрові застосування, що в свою чергу, підвищує рівень зацікавленості розроблюваним ігровим продуктом серед ігрової спільноти.

Метою створення розробки є розширення підходу до створення ігрових механік, а саме, впровадження бота-суперника, навченого з використанням нейронної мережі, для протидії реальному гравцю.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Вимоги до функціональних характеристик

4.1.1. Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

– взаємодія з користувацьким інтерфейсом;

– взаємодія з ігровим середовищем;

– взаємодія з ігровим ботом-суперником.

Розробку виконати на платформі Unity

4.1.2. Додаткові вимоги:

Не висуваються.

4.2. Вимоги до надійності

4.2.1. Передбачити контроль введення інформації.

4.2.2. Передбачити захист від некоректних дій користувача.

4.3. Умови експлуатації

4.3.1. Умови експлуатації згідно СанПін 2.2.2.542 – 96.

Не висуваються.

4.3.2. Обслуговування та обслуговуючий персонал.

Не висуваються.

4.4. Вимоги до складу і параметрів технічних засобів

4.4.1. Програмне забезпечення повинно функціонувати на обчислювальних машинах з ОС macOS.

4.4.2. Мінімальна конфігурація технічних засобів:

4.4.3. Тип процесору

4.4.4. Будь-який процесор що підтримує ОС macOS.

4.4.5. Об‘єм ОЗП

4.4.6. 200 Мб, або більше.

4.4.7. Пристрої введення

4.4.8. Оптичний та потоковий пристрої введення.

4.5. Вимоги до інформаційної та програмної сумісності

– Програмне забезпечення повинно працювати під управлінням операційної системи macOS.

4.6. Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7. Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8. Спеціальні вимоги

Згенерувати установочну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1. Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі

5.2. Програмне забезпечення повинно мати довідникову систему

5.3. У склад супроводжувальної документації повинні входити наступні документи:

5.3.1. Пояснювальна записка не менше ніж на 100 аркушах формату А4 (без додатків 5.3.2 - 5.3.6).

5.3.2. Технічне завдання

5.3.3. Керівництво користувача

5.3.4. Програма та методика тестування

5.4. Графічна частина повинна бути виконана на 3 листах формату А3, котрі включаються у якості графічних матеріалів до пояснювальної записки:

5.4.1. Схема структурна варіантів взаємодії

5.4.2. Схема структурна компонентів програмного забезпечення

5.4.3. Схема структурна класів програмного

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	10.03.2020	
2.	Розробка технічного завдання	15.03.2020	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	01.04.2020	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	15.04.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	15.05.2020	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	25.05.2020	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	30.05.2020	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проекту	30.05.2020	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	06.06.2020	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування описано в пояснювальній записці «КПІ.ІП-6312.045490.02.81», розділі 3 «Аналіз та тестування програмного забезпечення».

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
“___” _____ 2020 р.

ІГРОВЕ ЗАСТУВАННЯ З НАВЧАННЯМ БОТІВ-СУПЕРНИКІВ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Опис програми

КПІ.ІП-6312.045490.04.13

“ПОГОДЖЕНО”
Керівник проєкту:
_____ О.В. Ковтунець

Нормоконтроль:
_____ К.І. Ліщук

Виконавець:
_____А.В. Зозуля

Київ – 2020 року

Тексти програмного коду

Ігрове застування з навчанням ботів-суперників з використанням нейронних мереж

23 арк, 244 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2020

ApplicationContoller.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ApplicationContoller : MonoBehaviour
{
    public PlayerSpawner PlayerSpawner;
    public MazeSpawner MazeSpawner;
    private bool controllersInitialized;

    private GameManager gameManager;

    private void Awake()
    {
        Broadcaster.Instance.AddSubscription("GameStateManager." +
GameState.Spawning.ToString(), StartControllers);
        Broadcaster.Instance.AddSubscription("GlobalReset", ResetControllers);
        GameManager.Instance.AddGlobal("AgentReset", false);

        controllersInitialized = false;
    }

    private void Start()
    {
        GameManager.GameStateManager.SetState(GameState.Spawning);
        GameManager.Instance.AddGlobal("AgentReset", true);
    }

    public void ResetControllers()
    {
        if (controllersInitialized)
        {
            controllersInitialized = false;
            PlayerSpawner.Reset();
            MazeSpawner.Reset();
        }
    }

    public void StartControllers()
    {
        if (!controllersInitialized)
        {
            controllersInitialized = true;
            MazeSpawner.SpawnMaze();
            PlayerSpawner.SpawnPlayers();
        }
    }
}
```

Broadcaster.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using UnityEngine.Events;

public class Broadcaster : MonoBehaviourSingleton<Broadcaster>
{
    private Dictionary<string, UnityEvent> eventDictionary;
```

```
    protected override void SingletonAwakened()
    {
        eventDictionary = new Dictionary<string, UnityEvent>();
    }

    public void AddSubscription(string eventName, UnityAction listener)
    {
        UnityEvent thisEvent;
        if (eventDictionary.ContainsKey(eventName))
        {
            if (eventDictionary.TryGetValue(eventName, out thisEvent))
            {
                thisEvent.AddListener(listener);
            }
        }
        else
        {
            thisEvent = new UnityEvent();
            thisEvent.AddListener(listener);
            eventDictionary.Add(eventName, thisEvent);
        }
    }

    public void RemoveSubscription(string eventName, UnityAction listener)
    {
        if (eventDictionary.ContainsKey(eventName))
        {
            if (eventDictionary.TryGetValue(eventName, out UnityEvent thisEvent))
            {
                thisEvent.RemoveListener(listener);
            }
        }
    }

    public void BroadcastGlobalMessage(string eventName)
    {
        if (eventDictionary.ContainsKey(eventName))
        {
            if (eventDictionary.TryGetValue(eventName, out UnityEvent thisEvent))
            {
                thisEvent.Invoke();
            }
        }
    }
}
```

UIContoller.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class UIContoller : MonoBehaviour
{
    void Awake()
    {
        Broadcaster.Instance.AddSubscription("GameStateManager." +
GameState.InProgress.ToString(), OnStartGame);
```

```
        Broadcaster.Instance.AddSubscription("GameStateManager." +
GameState.Spawning.ToString(), HideAll);
    }

    void HideAll()
    {
        Broadcaster.Instance.BroadcastGlobalMessage("MainHUD.Hide");
    }

    void OnStartGame()
    {
        Broadcaster.Instance.BroadcastGlobalMessage("MainHUD.Show");
    }

    public void OnPauseClick()
    {
        Broadcaster.Instance.BroadcastGlobalMessage("MainHUD.Hide");
        GameManager.GameStateManager.SetState(GameState.Pause);
    }

    public void OnResumeClick()
    {
        GameManager.GameStateManager.SetState(GameState.InProgress);
    }

    public void OnQuitClick()
    {
        Application.Quit();
    }

    public void OnResetClick()
    {
        Broadcaster.Instance.BroadcastGlobalMessage("GlobalReset");
    }

    public void OnStartClick()
    {
        SceneManager.LoadScene("Maze3D");
    }

    public void OnSettingsClick()
    {

    }

    private void OnApplicationPause(bool pause)
    {
        if (pause)
        {
            OnPauseClick();
        }
    }
}
```

PlayerFollow.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerFollow : MonoBehaviour {
```

```
    public Transform PlayerTransform;

    public Vector3 _cameraOffset = Vector3.zero;

    [Range(0.01f, 1.0f)]
    public float SmoothFactor = 0.5f;

    public bool LookAtPlayer = false;

    public PlayerType PlayerType = PlayerType.Escaper;

                                                    // Use this for initialization
                                                    void Start ()
    {
        Broadcaster.Instance.AddSubscription("GameStateManager." +
GameState.Spawning.ToString(), SetValues);
                                                    }

                                                    // LateUpdate is called after
Update
                                                    void Update () {
        if (PlayerTransform != null)
        {
            Vector3 newPos = PlayerTransform.position + _cameraOffset;

            transform.position = Vector3.Slerp(transform.position, newPos,
SmoothFactor);

            if (LookAtPlayer)
                transform.LookAt(PlayerTransform);
        }
                                                    }

    void SetValues()
    {
        PlayerTransform = GameManager.PlayerManager.GetPlayer(PlayerType).transform;

        if (_cameraOffset == Vector3.zero)
            _cameraOffset = transform.position - PlayerTransform.position;
    }
}
```

PlayerControls3D.cs

```
using UnityEngine;

public class PlayerControls3D : MonoBehaviour
{
    public float Speed = 2;

    private Rigidbody componentRigidbody;

    private void Start()
    {
        componentRigidbody = GetComponent<Rigidbody>();
        Speed = GetComponent<PlayerEntity>().Speed;
    }

    private bool paused = true;
```

```
    private void Awake()
    {
        Broadcaster.Instance.AddSubscription("GameStateManager." +
GameState.Pause.ToString(), OnPaused);
        Broadcaster.Instance.AddSubscription("GameStateManager." +
GameState.InProgress.ToString(), InProgress);
    }

    private void OnPaused()
    {
        paused = true;
    }

    private void InProgress()
    {
        paused = false;
    }

    private void Update()
    {
        if (!paused)
        {
            var dirToGo = Vector3.zero;
            var rotateDir = Vector3.zero;
            if (Input.GetKey(KeyCode.D))
            {
                rotateDir = transform.right * 1f;
            }
            else
            if (Input.GetKey(KeyCode.A))
            {
                rotateDir = transform.right * -1f;
            }

            if (Input.GetKey(KeyCode.W))
            {
                dirToGo = transform.forward * 1f;
            }
            else
            if (Input.GetKey(KeyCode.S))
            {
                dirToGo = transform.forward * -1f;
            }

            componentRigidbody.AddForce((dirToGo + rotateDir).normalized * Speed,
ForceMode.VelocityChange);
        }
    }
}
```

RandomMazeWalkController.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RandomMazeWalkController : MonoBehaviour
{
    private float Speed = 1;
    private Rigidbody componentRigidbody;
```

```
    private MazeGeneratorCell targetCell;
    private MazeGeneratorCell previousCell;
    private Maze mazeEntity;
    private bool paused = false;

    private void Awake()
    {
        Broadcaster.Instance.AddSubscription("GameStateManager." +
GameState.Pause.ToString(), OnPaused);
        Broadcaster.Instance.AddSubscription("GameStateManager." +
GameState.InProgress.ToString(), InProgress);
    }

    private void OnPaused()
    {
        paused = true;
    }

    private void InProgress()
    {
        paused = false;
    }

    private void Start()
    {
        componentRigidbody = GetComponent<Rigidbody>();
        componentRigidbody.velocity = Vector2.zero;
        mazeEntity = GameManager.MazeManager.GetMazeEntity();

        var player = GetComponent<PlayerEntity>();
        Speed = player == null ? Speed : player.Speed;
    }

    void FixedUpdate()
    {
        if (!paused)
        {
            Vector3 velocity = componentRigidbody.velocity;

            if (targetCell == null)
            {
                targetCell = mazeEntity.GetClosestCell(transform.position);
                if (targetCell == null)
                {
                    Debug.LogAssertion("targetCell == null");
                    return;
                }
            }

            var minDistance = Math.Sqrt(Math.Pow(mazeEntity.CellData.Size.x, 2) +
Math.Pow(mazeEntity.CellData.Size.z, 2)) / 3;

            if (Vector3.Distance(mazeEntity.GetCellScreenPosition(targetCell),
transform.position) < minDistance)
            {
                var accesibleCells = mazeEntity.GetAccesibleCells(targetCell);
                accesibleCells.Remove(previousCell);
                if (accesibleCells.Count > 0)
                {
                    previousCell = targetCell;
```

```
                    targetCell = accesibleCells[UnityEngine.Random.Range(0,
accesibleCells.Count)];

                    Vector3 correctionVelocity =
(mazeEntity.GetCellScreenPosition(targetCell) -
mazeEntity.GetCellScreenPosition(previousCell)).normalized * Speed;
                    velocity += correctionVelocity;
                }
                else
                {
                    var tempCell = targetCell;
                    targetCell = previousCell;
                    previousCell = tempCell;
                }
            }
            else
            {
                velocity = (mazeEntity.GetCellScreenPosition(targetCell) -
transform.position).normalized * Speed;
            }

            Debug.DrawRay(transform.position, velocity * 20, Color.red);

            componentRigidbody.AddForce(velocity, ForceMode.VelocityChange);
        }
    }
}
```

PlayerSpawner.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerSpawner : MonoBehaviour
{
    public PlayerEntity SpawnPlayer(Vector3 position, PlayerData playerData)
    {
        position.y = playerData.prefab.transform.position.y;
        PlayerEntity player = Instantiate(playerData.prefab, position,
Quaternion.identity, gameObject.transform);

        if (null != player)
        {
            player.Type = playerData.Type;
            player.Speed = playerData.Speed;

            GameManager.PlayerManager.AddPlayer(ref player);
        }
        else
        {
            Debug.Assert(false, "player == null");
        }

        return player;
    }

    public void SpawnPlayers()
    {
        var mazeEntity = GameManager.MazeManager.GetMazeEntity();
        foreach (var player in GameManager.PlayerManager.GetSpawnList())
```

```
        {
            var cellPosition =
mazeEntity.GetCellScreenPosition(mazeEntity.GetRandomFreeCell());
            SpawnPlayer(cellPosition, player);
        }
    }

    public void Reset()
    {
        foreach (var player in GameManager.PlayerManager.GetPlayers())
        {
            try
            {
               Destroy(player.gameObject);
            }
            catch (System.Exception ex)
            {
                Debug.LogAssertion(ex.Message);
            }
        }
    }
}
```

Maze.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;

public class MazeGeneratorCell
{
    public int X;
    public int Y;

    public bool WallVertical = true;
    public bool WallHorizontal = true;

    public bool Visited = false;
    public int DistanceFromStart;

    public GameObject GameObject { get; set; }

    public Vector3 Position { get => GameObject.transform.position; }

    public int Distance = 0;
}

public enum MazeType
{
    Default,
    DefaultWithCycles
}

[CreateAssetMenu(menuName = "Mazes/Maze", fileName = "NewMazeEntity")]
public class Maze : ScriptableObject
{
    public MazeGeneratorCell[,] cells;
    public Vector2Int FinishPosition;
    public MazeGeneratorCell FinishCell { get; private set; }
    public int Width;
    public int Height;
```

```
    public CellData CellData;
    public MazeType Type;

    public Vector3 GetCellScreenPosition(MazeGeneratorCell cell)
    {
        return GetIndexPosition(cell.X, cell.Y);
    }

    public MazeGeneratorCell GetRandomCell()
    {
        return cells[(uint)UnityEngine.Random.Range(0, Width - 1),
(uint)UnityEngine.Random.Range(0, Height - 1)];
    }

    public MazeGeneratorCell GetRandomFreeCell()
    {
        MazeGeneratorCell randomCell;
        bool isEmpty = true;
        uint counter = 0;
        do
        {
            randomCell = GetRandomCell();
            foreach (var player in GameManager.PlayerManager.GetPlayers())
            {
                if (randomCell == GetClosestCell(player.gameObject.transform.position))
                {
                    isEmpty = false;
                    break;
                }
            }
            counter++;
        } while (!isEmpty && counter < Height * Width);

        return randomCell;
    }

    public List<MazeGeneratorCell> GetAccesibleCells(MazeGeneratorCell cell)
    {
        List<MazeGeneratorCell> accessibleCells = new List<MazeGeneratorCell>();
        var x = cell.X;
        var y = cell.Y;

        if (cell.X > 0 && !cell.WallVertical)
        {
            accessibleCells.Add(cells[x - 1, y]);
        }

        if (y > 0 && !cell.WallHorizontal)
        {
            accessibleCells.Add(cells[x, y - 1]);
        }

        if (x + 1 < Width && !cells[x + 1, y].WallVertical)
        {
            accessibleCells.Add(cells[x + 1, y]);
        }

        if (y + 1 < Height && !cells[x, y + 1].WallHorizontal)
        {
            accessibleCells.Add(cells[x, y + 1]);
```

```
        }

        return accessibleCells;
    }

    public Vector3 GetIndexPosition(int x, int y, bool ignoringSize = false)
    {
        if (!(x >= 0 && x < Width && y >= 0 && y < Height))
        {
            Debug.Assert(false);
            return Vector2.zero;
        }

        Vector3 pos3D = cells[x, y].Position;
        Vector3 sizeOffset = (ignoringSize ? 0f : 0.5f) * CellData.Size;

        return sizeOffset + pos3D;
    }

    public MazeGeneratorCell GetClosestCell(Vector3 position)
    {
        MazeGeneratorCell closestCell = null;

        if (IsInsideMaze(position))
        {
            Vector3 index = (position - GetIndexPosition(0, 0, true));

            index.x /= CellData.Size.x;
            index.y /= CellData.Size.y;
            index.z /= CellData.Size.z;

            Vector2Int intIndex = new Vector2Int((int)Math.Floor(index.x),
(int)Math.Floor(index.z));
            closestCell = cells[intIndex.x, intIndex.y];

            if (closestCell.X == Width - 1 || closestCell.Y == Height - 1)
            {
                Debug.LogError("Cell is outside maze");
            }
        }
        return closestCell;
    }

    private bool IsInsideMaze(Vector3 position)
    {
        var leftBottomCell = cells[0, 0].Position;
        var rigthTopCell = cells[Width - 1, Height - 1].Position;

        bool isInside = true;

        if (position.x <= leftBottomCell.x || position.z <= leftBottomCell.z)
        {
            isInside = false;
        }
        else if (position.x >= rigthTopCell.x || position.y >= rigthTopCell.z)
        {
            isInside = false;
        }

        return isInside;
```

```
    }

    public MazeGeneratorCell GetFinishCell()
    {
        if (FinishCell == null)
        {
            int side = UnityEngine.Random.Range(0, 4);
            switch (side)
            {
                case 0: FinishCell = cells[0, (int)UnityEngine.Random.Range(0, Height -
1)]; break;
                case 1: FinishCell = cells[(int)UnityEngine.Random.Range(0, Width - 1),
Height - 2]; break;
                case 2: FinishCell = cells[Width - 2, (int)UnityEngine.Random.Range(0,
Height - 1)]; break;
                case 3: FinishCell = cells[(int)UnityEngine.Random.Range(0, Width - 1),
0]; break;
            }
        }

        return FinishCell;
    }

    public Vector3 NormalizedPosition(Vector3 position)
    {
        var offset = cells[Width - 1, Height - 1].Position - cells[0, 0].Position;
        var normalizedPosition = position;

        normalizedPosition.x /= offset.x;
        normalizedPosition.y /= offset.y;
        normalizedPosition.z /= offset.z;

        return normalizedPosition;
    }

    public void Reset()
    {
        cells = null;
        FinishCell = null;
    }

    public int DistanceBetweenCells(MazeGeneratorCell start, MazeGeneratorCell target)
    {
        if (start == target)
        {
            return 0;
        }

        Wave(start, target, 1);

        int distance = -1;
        if (target.Distance > 0)
        {
            distance = BackWave(target, start, 0);
        }

        for (uint x = 0; x < cells.GetLength(0); x++)
        {
            for (uint y = 0; y < cells.GetLength(1); y++)
            {
```

```
                cells[x, y].Distance = 0;
            }
        }

        return distance;
    }

    private void Wave(MazeGeneratorCell currentCell, MazeGeneratorCell target, int
distance)
    {
        currentCell.Distance = distance;

        if (currentCell == target)
        {
            return;
        }

        var closestCells = GetAccesibleCells(currentCell);
        foreach (var cell in closestCells)
        {
            if (cell.Distance == 0)
            {
                Wave(cell, target, distance + 1);
            }
        }
    }

    private int BackWave(MazeGeneratorCell currentCell, MazeGeneratorCell target, int
distanceCumulative)
    {
        if (currentCell == target)
        {
            return distanceCumulative;
        }

        var closestCells = GetAccesibleCells(currentCell);
        if (closestCells.Count > 0)
        {
            MazeGeneratorCell minCell = currentCell;
            foreach (var cell in closestCells)
            {
                if (cell.Distance > 0 && cell.Distance < minCell.Distance)
                {
                    minCell = cell;
                }
            }

            if (minCell != currentCell)
            {
                return BackWave(minCell, target, distanceCumulative + 1);
            }
            else
            {
                return distanceCumulative;
            }
        }
        else
        {
            return distanceCumulative;
        }
```

```
    }
}
```

MazeGenerator.cs

```
using System.Collections.Generic;

public class MazeGenerator
{
    public MazeGeneratorCell[,] GenerateMaze(Maze currentMaze)
    {
        currentMaze.cells = new MazeGeneratorCell[currentMaze.Width,
currentMaze.Height];

        for (int x = 0; x < currentMaze.cells.GetLength(0); x++)
        {
            for (int y = 0; y < currentMaze.cells.GetLength(1); y++)
            {
                currentMaze.cells[x, y] = new MazeGeneratorCell { X = x, Y = y };
            }
        }

        for (int x = 0; x < currentMaze.cells.GetLength(0); x++)
        {
            currentMaze.cells[x, currentMaze.Height - 1].WallVertical = false;
        }

        for (int y = 0; y < currentMaze.cells.GetLength(1); y++)
        {
            currentMaze.cells[currentMaze.Width - 1, y].WallHorizontal = false;
        }

        switch(currentMaze.Type)
        {
            case MazeType.Default:
                BacktraceMaze(currentMaze.cells, currentMaze.cells[0, 0], false);
break;
            case MazeType.DefaultWithCycles:
                BacktraceMaze(currentMaze.cells, currentMaze.cells[0, 0], true); break;
        }

        return currentMaze.cells;
    }

    private void BacktraceMaze(MazeGeneratorCell[,] cells, MazeGeneratorCell startCell,
bool withCycles)
    {
        MazeGeneratorCell currentCell = startCell;
        currentCell.Visited = true;

        int Width = cells.GetLength(0);
        int Height = cells.GetLength(1);

        Stack<MazeGeneratorCell> visitedCells = new Stack<MazeGeneratorCell>();

        do
        {
            List<MazeGeneratorCell> unvisitedNeighbours = new
List<MazeGeneratorCell>();

            int x = currentCell.X;
```

```
            int y = currentCell.Y;

            if (x > 0 && !cells[x - 1, y].Visited)
            {
                unvisitedNeighbours.Add(cells[x - 1, y]);
            }

            if (y > 0 && !cells[x, y - 1].Visited)
            {
                unvisitedNeighbours.Add(cells[x, y - 1]);
            }

            if (x < Width - 2 && !cells[x + 1, y].Visited)
            {
                unvisitedNeighbours.Add(cells[x + 1, y]);
            }

            if (y < Height - 2 && !cells[x, y + 1].Visited)
            {
                unvisitedNeighbours.Add(cells[x, y + 1]);
            }


            if (unvisitedNeighbours.Count > 0)
            {
                var wallsToRemoveCount = 1;
                if (unvisitedNeighbours.Count >= 3 && withCycles)
                {
                    wallsToRemoveCount = UnityEngine.Random.Range(1,
unvisitedNeighbours.Count);
                }

                for (int i = 0; i < wallsToRemoveCount; i++)
                {
                    MazeGeneratorCell chosen =
unvisitedNeighbours[UnityEngine.Random.Range(0, unvisitedNeighbours.Count)];

                    RemoveWall(currentCell, chosen);

                    chosen.Visited = true;
                    visitedCells.Push(chosen);

                    unvisitedNeighbours.Remove(chosen);

                    if (i < wallsToRemoveCount - 1)
                    {
                        BacktraceMaze(cells, chosen, withCycles);
                    }
                    else
                    {
                        currentCell = chosen;
                    }
                }
            }
            else
            {
                if (visitedCells.Count > 0)
                    currentCell = visitedCells.Pop();
            }
```

```
        } while (visitedCells.Count > 0);
    }

    private void RemoveWall(MazeGeneratorCell a, MazeGeneratorCell b)
    {
        if (a.X == b.X)
        {
            if (a.Y > b.Y)
            {
                a.WallHorizontal = false;
            }
            else
            {
                b.WallHorizontal = false;
            }
        }
        else
        {
            if (a.X > b.X)
            {
                a.WallVertical = false;
            }
            else
            {
                b.WallVertical = false;
            }
        }
    }
}
```

MazeManager.cs

```
using UnityEngine;
using System.Collections.Generic;
using Unity.MLAgents;

public class MazeManager
{
    private List<Maze> mazeEntities;
    private Maze currentMaze;
    private MazeGenerator mazeGenerator;

    public MazeManager()
    {
        mazeGenerator = new MazeGenerator();
        mazeEntities = ScriptableObjectsController.Instance.GetMazeEntities();
    }

    public Maze GetMazeEntity()
    {
        if (currentMaze == null)
        {
            int index =
Mathf.FloorToInt(Academy.Instance.EnvironmentParameters.GetWithDefault("maze", -1f));
            if (index > 0)
            {
                currentMaze = mazeEntities[index - 1];
            }
            else
            {
                currentMaze = mazeEntities[Random.Range(0, mazeEntities.Count)];
```

```
            }
            mazeGenerator.GenerateMaze(currentMaze);
        }

        return currentMaze;
    }

    public void Reset()
    {
        currentMaze = null;
        foreach (var maze in mazeEntities)
        {
            maze.Reset();
        }
    }

    public Vector3 GetMazeCenterPosition()
    {
        return currentMaze.GetIndexPosition(currentMaze.Width / 2 - 1,
currentMaze.Height / 2 - 1);
    }
}
```

PlayerManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerManager
{
    private List<PlayerEntity> Players;
    private List<PlayerData> SpawnList;

    public PlayerManager()
    {
        Players = new List<PlayerEntity>();
        SpawnList = ScriptableObjectsController.Instance.GetPlayersData();
    }

    public List<PlayerEntity> GetPlayers()
    {
        return Players;
    }

    public PlayerEntity GetPlayer(PlayerType type)
    {
        foreach (var player in Players)
        {
            if (player.Type == type)
            {
                return player;
            }
        }

        return null;
    }

    public bool RemovePlayer(PlayerEntity player)
    {
```

```
            return Players.Remove(player);
        }

        public List<PlayerData> GetSpawnList()
        {
            return SpawnList;
        }

        public void AddPlayer(ref PlayerEntity player)
        {
            if (player.Type == PlayerType.Escaper)
            {
                var mainPlayer = GetPlayer(PlayerType.Escaper);
                if (mainPlayer != null && mainPlayer != player)
                {
                    Debug.LogAssertion("Main player already exists");
                }
            }

            Players.Add(player);
        }

        public void Reset()
        {
            if (Players != null)
            {
                Players.Clear();
            }
        }
    }
```

PersecutorAgent.cs

```
using System.Collections;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;

public class PersecutorAgent : Agent
{
    public bool useVectorObs;
    Rigidbody m_AgentRb;
    PersecutorAgentSettings m_HallwaySettings;
    private PlayerEntity target;

    private float startDistance;
    private int currentDistance;
    private float catchingDistance;

    public float wallHitReward = -1f;
    public float losingDistanceReward = 0.1f;
    public float growingDistanceReward = -0.05f;

    private bool paused = true;

    private void Start()
    {
        //Broadcaster.Instance.AddSubscription("MainPlayer.OnFinish", RegisterLose);
        //Broadcaster.Instance.AddSubscription("MainPlayer.OnCatched", RegisterWin);
```

```
        Broadcaster.Instance.AddSubscription("GameStateManager." +
GameState.Pause.ToString(), OnPaused);
        Broadcaster.Instance.AddSubscription("GameStateManager." +
GameState.InProgress.ToString(), InProgress);
    }

    private void OnPaused()
    {
        paused = true;
    }

    private void InProgress()
    {
        paused = false;
    }

    public override void Initialize()
    {
        m_HallwaySettings = FindObjectOfType<PersecutorAgentSettings>();
        m_AgentRb = GetComponent<Rigidbody>();
        catchingDistance =
Academy.Instance.EnvironmentParameters.GetWithDefault("catching_distance", 0f);
        target = GameManager.PlayerManager.GetPlayer(PlayerType.Escaper);

        currentDistance = CallculateDistance();
        startDistance = currentDistance;

        GameManager.Instance.AddGlobal("AgentReset", false);
    }

    public override void OnEpisodeBegin()
    {
        if (GameManager.Instance.GetValue<bool>("AgentReset"))
        {
            Broadcaster.Instance.BroadcastGlobalMessage("GlobalReset");
        }
        else
        {
            GameManager.Instance.AddGlobal("AgentReset", true);
        }
    }

    public override void CollectObservations(VectorSensor sensor)
    {
        if (useVectorObs)
        {
            if (startDistance != 0f)
            {
                sensor.AddObservation(currentDistance / startDistance);
            }
            sensor.AddObservation(StepCount / (float)MaxStep);
        }
    }

    public void MoveAgent(float[] act)
    {
        if (!paused)
        {
            var dirToGo = Vector3.zero;
            var rotateDir = Vector3.zero;
```

```
            var actionMove = Mathf.FloorToInt(act[0]);
            switch (actionMove)
            {
                case 1:
                    dirToGo = transform.forward * 1f;
                    break;
                case 2:
                    dirToGo = transform.forward * -1f;
                    break;
            }

            var actionRotate = Mathf.FloorToInt(act[1]);
            switch (actionRotate)
            {
                case 1:
                    rotateDir = transform.right * 1f;
                    break;
                case 2:
                    rotateDir = transform.right * -1f;
                    break;
            }

            m_AgentRb.AddForce((dirToGo + rotateDir).normalized *
m_HallwaySettings.agentRunSpeed, ForceMode.VelocityChange);
        }
    }

    public override void OnActionReceived(float[] vectorAction)
    {
        if (!paused)
        {
            AddReward(-1f / MaxStep);
            MoveAgent(vectorAction);

            int distanceToTarget = CallculateDistance();

            if (distanceToTarget >= currentDistance)
            {
                AddReward(growingDistanceReward * StepCount / MaxStep);
            }
            else
            {
                AddReward(losingDistanceReward * (MaxStep - StepCount) / MaxStep);
            }

            currentDistance = distanceToTarget;
        }
    }

    void OnCollisionEnter(Collision col)
    {
        if (col.gameObject.CompareTag("wall"))
        {
            AddReward(wallHitReward);
        }
    }

    public override void Heuristic(float[] actionsOut)
    {
```

```
            actionsOut[0] = 0;
            actionsOut[1] = 0;
            if (Input.GetKey(KeyCode.D))
            {
                actionsOut[1] = 1;
            }
            else
            if (Input.GetKey(KeyCode.A))
            {
                actionsOut[1] = 2;
            }

            if (Input.GetKey(KeyCode.W))
            {
                actionsOut[0] = 1;
            }
            else
            if (Input.GetKey(KeyCode.S))
            {
                actionsOut[0] = 2;
            }
        }

        private void FixedUpdate()
        {
            // Reached target
            if (Vector3.Distance(transform.position, target.transform.position) <
    catchingDistance && !paused)
            {
                RegisterWin();
            }
        }

        private void RegisterWin()
        {
            if (GameManager.GameStateManager.CurrentGameState == GameState.InProgress)
            {
                SetReward(1.0f);
                EndEpisodeAndReset();
            }
        }

        public void RegisterLose()
        {
            if (GameManager.GameStateManager.CurrentGameState == GameState.InProgress)
            {
                SetReward(-1.0f);
                EndEpisodeAndReset();
            }
        }

        public void EndEpisodeAndReset()
        {
            Debug.Log("AreaReset");
            GameManager.Instance.AddGlobal("AgentReset", true);
            EndEpisode();
        }

        private int CallculateDistance()
        {
```

```
            var maze = GameManager.MazeManager.GetMazeEntity();
            var targetCell = maze.GetClosestCell(target.transform.position);
            var agentCell = maze.GetClosestCell(transform.position);

            var distance = maze.DistanceBetweenCells(agentCell, targetCell);

            return distance;
        }
    }
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ІГРОВЕ ЗАСТУВАННЯ З НАВЧАННЯМ БОТІВ-СУПЕРНИКІВ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Програма та методика тестування

КПІ.ІП-6312.045490.05.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.В. Ковтунець

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____А.В. Зозуля

Київ – 2020 року

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Ігрове застосування, яке являє собою програму, розроблену з використанням ігрового рушія Unity з використанням технології ML-Agents.

2 МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

– надійність – здатність програмного забезпечення виконувати задачі протягом встановленого проміжку часу або вказану кількість операцій;

– функціональність – визначається здатністю програмного забезпечення вирішувати задачі, що відповідають потребам користувача;

– зручність використання – можливість легкого розуміння, вивчення, використання та привабливість для користувача;

– забезпечення належного рівня безпеки даних;

– зручність взаємодії з ігровим застосуванням;

– відповідність дизайну вимогам Технічного завданняю.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

– функціональне тестування, зокрема на рівні Critical path test (базове тестування);

– тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування);

– тестування інтерфейсу.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами інструментарію SpecFlow.

Працездатність ігрового застосування перевіряється шляхом:

– динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;

– динамічного ручного тестування на відповідність функціональним вимогам;

– статичного тестування коду.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ІГРОВЕ ЗАСТУВАННЯ З НАВЧАННЯМ БОТІВ-СУПЕРНИКІВ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Керівництво користувача

КПІ.ІП-6312.045490.06.34

“ПОГОДЖЕНО”

Керівник проекту:

_____ Ковтунець О.В.

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ Зозуля А.В.

Київ – 2020 року

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Ігрове застосування з навчанням ботів-суперників з використанням нейронних мереж – це готовий програмний продукт з інтегрованими агентами на основі навчених моделей нейронних мереж.

Ігрове застосування написано з використанням ігрового рушія Unity та мови програмування C# для простого встановлення на будь-який популярний вид пристроїв, або простої інтеграції його модулів в інше ігрове застосування Навчена модель бота-суперника в 90% ігрових партій виграє суперника, який грає випадковим чином.

ПІДГОТОВКА ДО РОБОТИ

Для запуску ігрового застосування необхідно завантажити його проект з сторінки Releases його GitHub репозиторію та розархівувати на девайсі гравця під операційною системою Mac OS.

ВЗАЄМОДІЯ З ІГРОВИМ ЗАСТОСУВАННЯМ

Після встановлення потрібно запустити виконуваний файл game.app. У разі невдалого встановлення або виникнення будь-якої помилки при старті застосування сповістить користувача про помилку та зупинить свою роботу.

Після старту ігрового застосування, гравець має можливість взаємодіяти з ним через оптичний та потоковий пристрої введення. Стартовий екран з елементами користувацького інтерфейсу зображено на рисунку 3.1 – Графічний інтерфейс користувача.

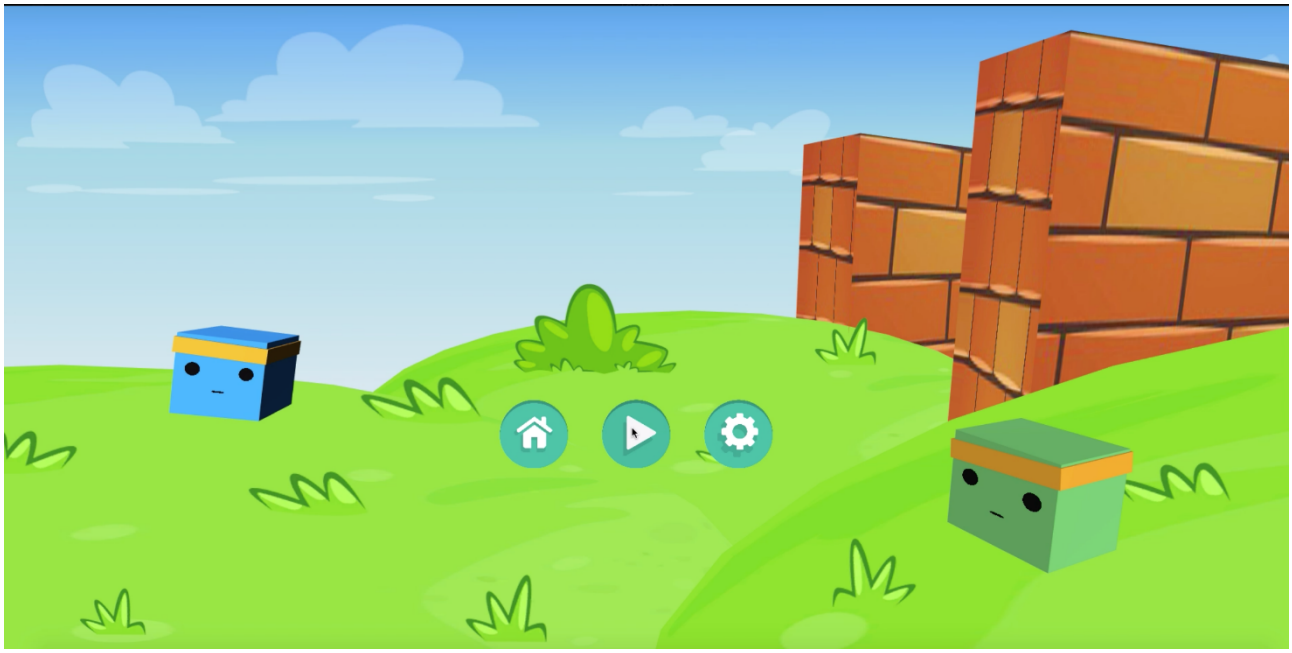


Рисунок 3.1 – Графічний інтерфейс користувача

Взаємодія з ігровим середовищем відбувається шляхом натискання контролюючих клавіш “W”, “A”, “S”, “D” на потоковому пристрої введення гравця та призначені для пересування ігрового персонажа гравця в просторі ігрового середовища вперед, ліворуч, назад та праворуч відповідно. Ігрове середовище являє собою скінченний лабіринт, що має комірку з полем фінішу та два ігрових персонажі гравця та бота-суперника відповідно. Ігрове середовище представлено на рисунку 3.2 – Приклад стану ігрового середовища.

Рисунок 3.2 – Приклад стану ігрового середовища

Суть ігрового процесу полягає в униканні зіткнень з ботом-суперником та русі лабіринтом до фінішного поля. В цей час бот-суперник намагається перешкоджати руху гравця та збирає дані для навчання в ігровому епізоді.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ІГРОВЕ ЗАСТОСУВАННЯ З НАВЧАННЯМ БОТІВ-СУПЕРНИКІВ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Графічний матеріал

КПІ.ІП-6312.045490.06.99.СС

“ПОГОДЖЕНО”

Керівник проекту:

_____О.В. Ковтунець

Нормоконтроль:

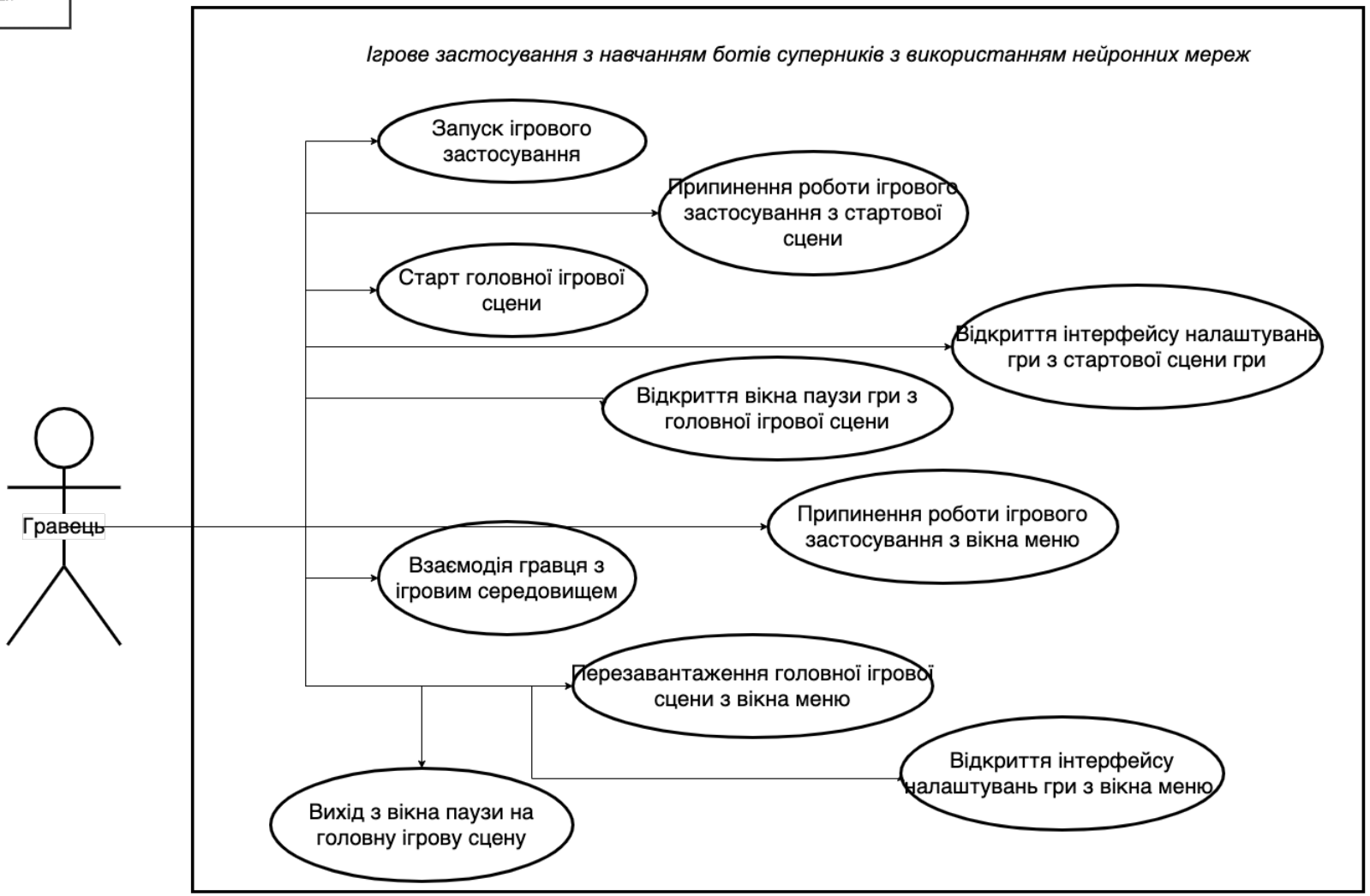
_____К.І. Ліщук

Виконавець:

_____А.В. Зозуля

Київ – 2020 року

КПІ.ІП-6312.045490.07.99.СС



					КПІ.ІП-6312.045490.07.99.СС	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використань	Літера / Маса / Масштаб
Розробив		Зозуля А.В.				
Перевірив		Ковтунець О.В.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Ліщук К.І.			Ігрове застосування з навчанням ботів суперників з використанням нейронних мереж	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63
Затвердив		Ковтунець О.В.				

КПІ.ІП-6312.045490.07.99.СС

					КПІ.ІП-6312.045490.07.99.СС	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення	Літера / Маса / Масштаб
Розробив		Зозуля А.В.				
Перевірив		Ковтунець О.В.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Ліщук К.І.			Ігрове застосування з навчанням ботів суперників з використанням нейронних мереж	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63
Затвердив		Ковтунець О.В.				

КПІ.ІП-6312.045490.07.99 СС

					КПІ.ІП-6312.045490.07.99.СС	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів програмного забезпечення	Літера / Маса / Масштаб
Розробив		Зозуля А.В.				
Перевірив		Ковтунець О.В.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Лящук К.І.			Ігрове застосування з навчанням ботів суперників з використанням нейронних мереж	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63
Затвердив		Ковтунець О.В.				